Registration No.

23320

-*Technical Report*-

APPROXIMATING SMOOTH STEP FUNCTIONS USING PARTIAL FOURIER SERIES SUMS

*****

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

*****

September 2012

U.S. Army Tank Automotive Research, Development, and Engineering Center
Detroit Arsenal
Warren, Michigan 48397-5000

TARDEC Technical Report No. 23320
September 2012

APPROXIMATING SMOOTH STEP FUNCTIONS USING PARTIAL FOURIER SERIES SUMS

W. Bylsma
Dynamics and Structures
U.S. Army Research, Development and Engineering Command (RDECOM)
U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC)
Detroit Arsenal
ATTN: RDTA-RS/MS157
6501 East 11 Mile Road
Warren, Michigan 48397-5000

* * * * *

* * * * *

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
09-17-2012	TECHNICAL	2012

APPROXIMATING SMOOTH STEP FUNCTIONS USING PARTIAL FOURIER SERIES SUMS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
WESLEY BYLSMA

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DYNAMICS AND STRUCTURES-US ARMY RDECOM/TARDEC
ATTN: RDTA-RS/MS157
6501 E 11 MILE RD
WARREN, MI 48397-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

23320

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Statement A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report documents several continuous functions that can approximate a discrete step function. The functions considered include the cosine, cubic polynomial, bezier polynomial, and hyperbolic tangent. Based on the error condition used, the Hyperbolic Tangent (Tanh) and Cosine functions provide the "best" approximations depending on the degree of smoothness desired.

15. SUBJECT TERMS
smooth step, partial sum, Fourier series, cosine, cubic polynomial, bezier polynomial, hyperbolic tangent

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	Unclassified	36	Wesley Bylsma
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** *(include area code)* 586-282-4104

CONTENTS (BY TITLE)

APPROXIMATING SMOOTH STEP FUNCTIONS USING PARTIAL FOURIER SERIES SUMS

TARDEC Technical Report No. 23320
September 2012

1.0 INTRODUCTION

This report documents several continuous functions that can approximate the transition between different constant levels of a signal or step function. The desire to avoid abrupt changes in signal levels will be demonstrated. Figure 1 illustrates the problem where a continuous transition is sought from level L_0 to L_1 between -ε and $+\varepsilon$. While this has been done in [1] the affects on the signal have never been clearly demonstrated.

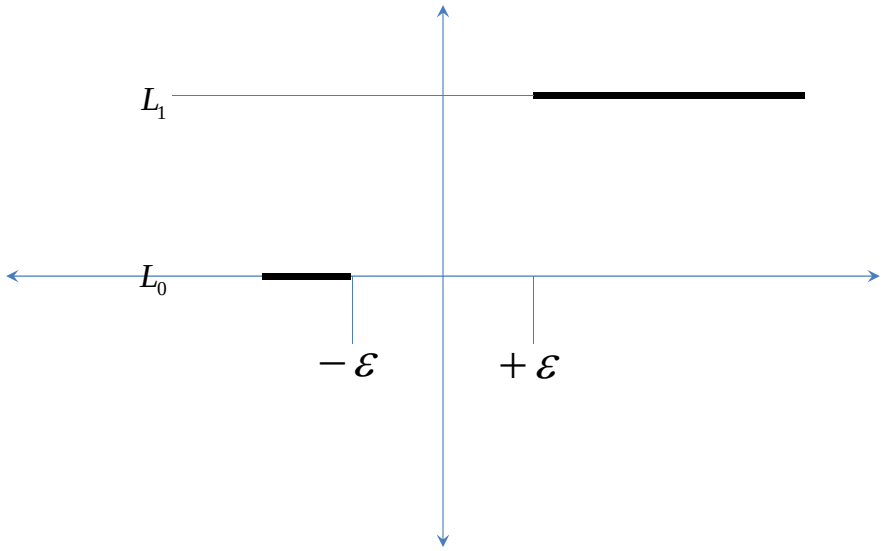


Figure 1 – Signal Levels and Transition Space

The continuous functions that will be considered are the cosine, cubic polynomial, Bezier polynomial, and hyperbolic tangent.

2.0 FOURIER REPRESENTATION OF SIGNALS

Before analyzing the individual functions mentioned in Section 1.0, a brief review of Fourier theory is appropriate. For periodic signals the Fourier representation is

$$x(t) = \sum_{n=0}^{\infty} (a_n cos\, n\omega_0 t + b_n sin\, nw_0 t) \tag{1}$$

where (T is the length of one period)

$$\omega_0 = \frac{2\pi}{T} \tag{2}$$

and noting that the summation (to infinity) will be relaxed to a partial sum (a large number of terms) that closely approximates the original signal. As outlined in [2], with the identities

$$\sin\theta = \frac{-j}{2}\left(e^{j\theta} - e^{-j\theta}\right) \tag{3}$$

$$\cos\theta = \frac{1}{2}\left(e^{j\theta} + e^{-j\theta}\right) \tag{4}$$

a complex number representation of the Fourier sum is defined as

$$x(t) = \sum_{n=-\infty}^{\infty} c_n e^{jnw_0 t} \tag{5}$$

with

$$c_n = \frac{a_n - jb_n}{2} \tag{6}$$

$$c_n^* = c_{-n} = \frac{a_n + jb_n}{2} \tag{7}$$

$$c_0 = a_0. \tag{8}$$

For even functions

$$a_n = 2c_n, b_n = 0.$$

From (5) ([2], pg. 21) the Fourier coefficients can be obtained from the original signal as

$$c_m = \frac{1}{T}\int_{-T/2}^{T/2} x(t)e^{-jm\omega_0 t}dt \tag{9}$$

For the periodic square pulse shown in Figure 2 the Fourier coefficients are

$$c_n = \frac{1}{T}\int_{-\tau/2}^{\tau/2} e^{\frac{-j2\pi nt}{T}}dt = \frac{1}{-j2\pi n}\left(e^{\frac{-j2\pi nt}{T}}\right)\Big|_{-\tau/2}^{\tau/2} = \frac{\tau}{T}sinc\ n\frac{\tau}{T} = d\ sinc\ nd \tag{10}$$

where $d = \tau/T$. This will be used as the test function to determine the affects the continuous step functions have on the Fourier representation of the signal and its approximation in the interval of interest.

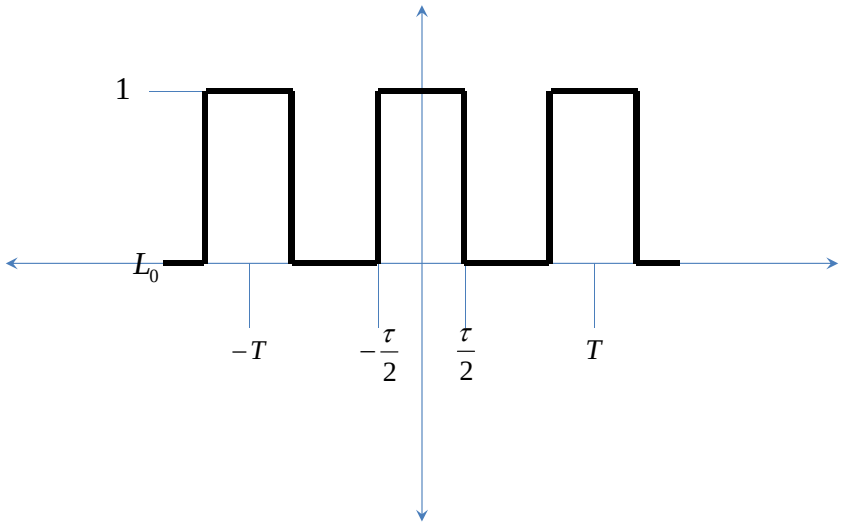


Figure 2 – Square Pulse Train Definition (ref. [2])

3.0 DIRECT VERSUS NUMERICAL INTEGRATION

The four functions mentioned in Section 1.0 increase the complexity of the integration in (10) needed to obtain a closed form solution for the Fourier coefficients. Here we pause to investigate the feasibility of obtaining the coefficients by numerical integration. Using MATLAB (and its notation, see [3]) to define the resulting Fourier coefficients in (10) as

Closed Form

```
cn=(2)*d*sinc(n*d)
```

Direct

```
di=(2)*real( (exp(-j*2*pi()*n*(tau/2)/T)/(-j*2*pi()*n) ) - ( exp(-
             j*2*pi()*n*(-tau/2)/T)/(-j*2*pi()*n) ) )
```

Numerical

```
ni=(2)*1/T*real( trapz(exp(-j*2*pi()*n.*t/T)) )*dt
```

the maximum error for direct and numerical integration (using trapezoidal integration (`trapz`)) is shown in Figure 3. The maximum error is defined as the maximum difference between the closed form solution for a 50 term partial Fourier series sum at each time step size. The initial maximum error was set to the machine precision epsilon (eps = 2.2204e-16). The results indicate that with small time steps, less than 0.001 (1e-3), numerical integration is an acceptable method to obtain the Fourier coefficients. Time steps less than 1-e6 increase the computational time and memory requirements significantly. For details of the computation see Appendix A.1 – Integration Differences.

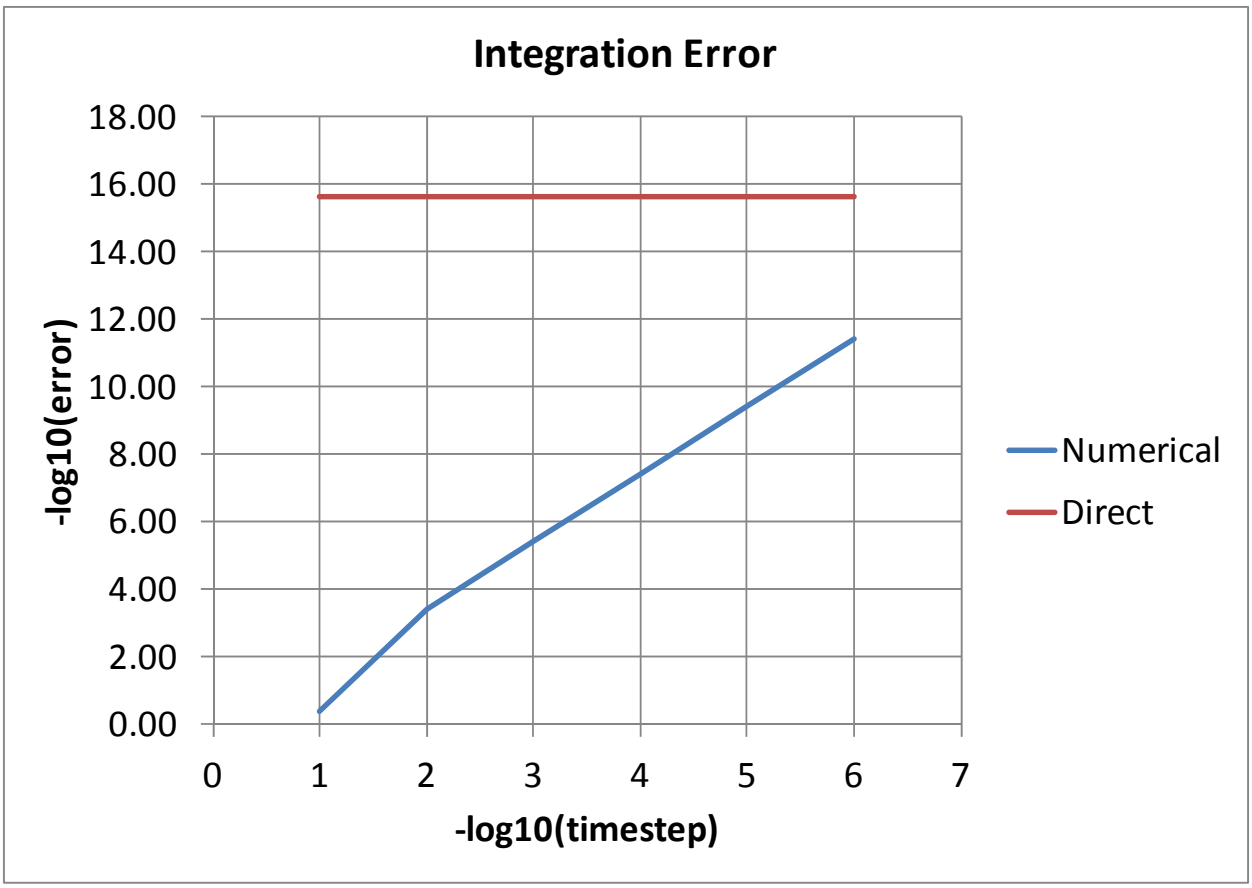


Figure 3 – Fourier Coefficient Integration Error Comparison - Numerical and Direct

4.0 ERROR CRITERIA

In considering the "best" continuous function approximation to the discrete step, two points need to be considered: 1) Smoothness and 2) Accuracy. The trade-off between the two is reflected in a fewer number of Fourier series terms needed to reach a "good" approximation for smooth signals versus the reduced accuracy the smooth signal represents the non-smooth signal (a discrete step in this case) that is being approximated. In order to capture this three error terms are defined

$$e1 = discrete\ step\ partial\ fourier\ series\ sum\ (N) - \ discrete\ step$$

$$e2 = smooth\ step\ partial\ fourier\ series\ sum\ (N) - \ smooth\ step\ (epsilon)$$

$$e3 = smooth\ step\ (epsilon) - discrete\ step \tag{11}$$

Note the dependencies in the error on---N, the number of Fourier series terms, and epsilon, the distance allowed for smoothing the discrete transition. In short, e1 and e2 represent the error between what is trying to be approximated with a partial Fourier series sum and e3 represents the difference between the discrete and smoothed representations before each is approximated with a partial Fourier series sum.

With these definitions, the criteria for selecting the “best” smooth approximation is defined when

$$e2(N,\varepsilon) + e3(\varepsilon) < e1(N) \tag{12}$$

expressed as a percentage of the pulse area. This criteria gives an upper bound, dependent on N and epsilon.

Figures 4 and 5 show examples of the smooth step functions for epsilon at 10% and 50% of tau respectively. For the examples in this report the parameters (period length, pulse width, number of terms) for Figure 2 are

$$T = 25, \tau = 12.5, N = 150.$$

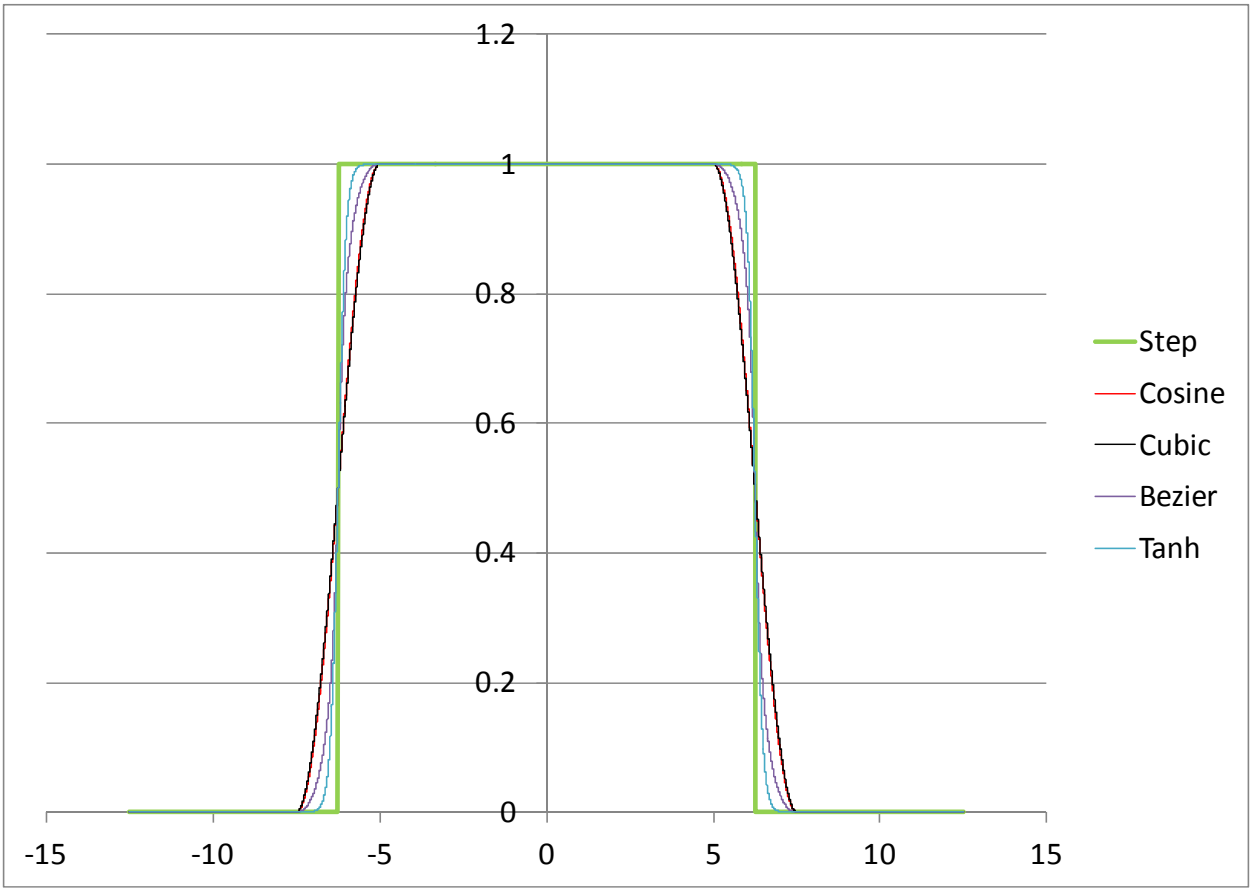


Figure 4 – Smooth Step Functions with epsilon equal to 10% of the pulse width (tau).

Figure 6 shows a zoomed in right side of Figure 5.

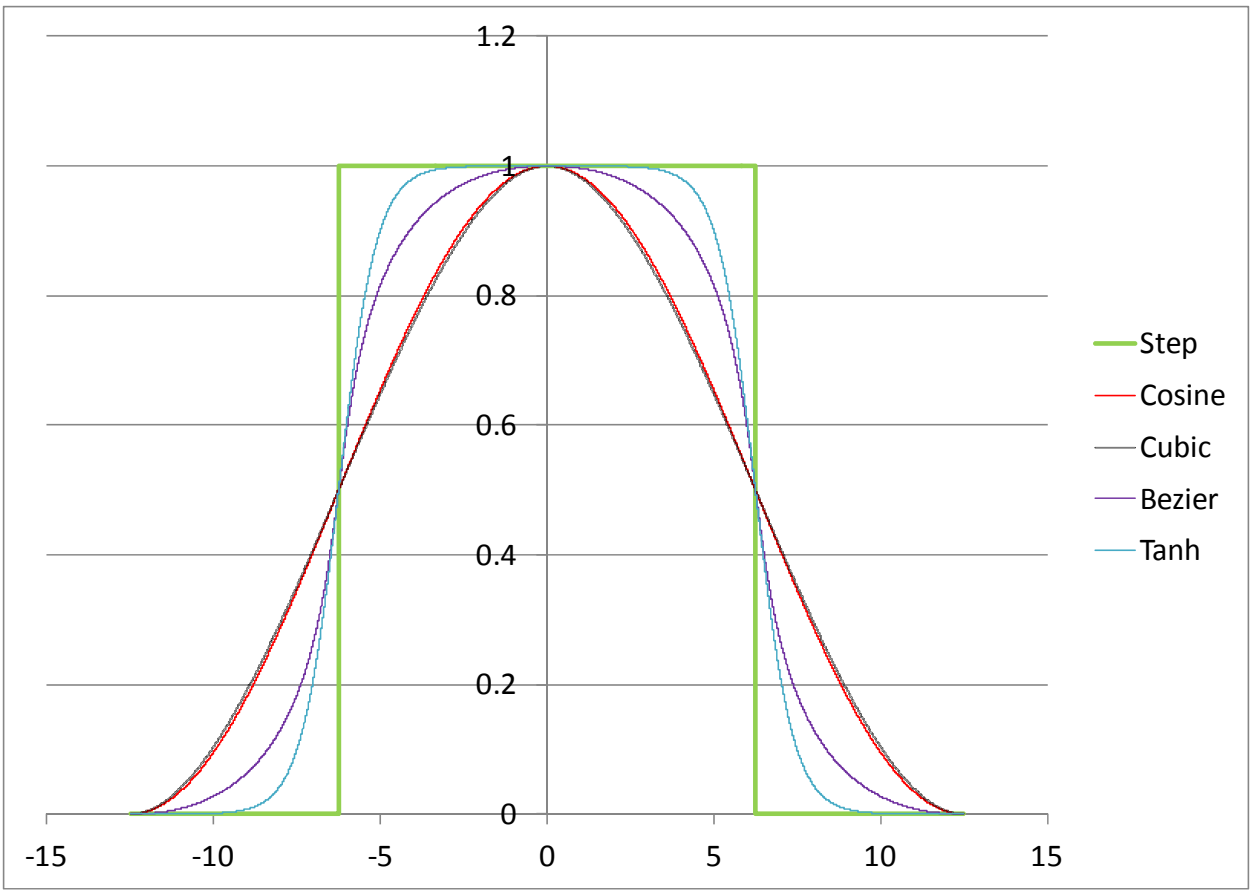


Figure 5 – Smooth Step Functions with epsilon equal to 50% of the pulse width (tau).

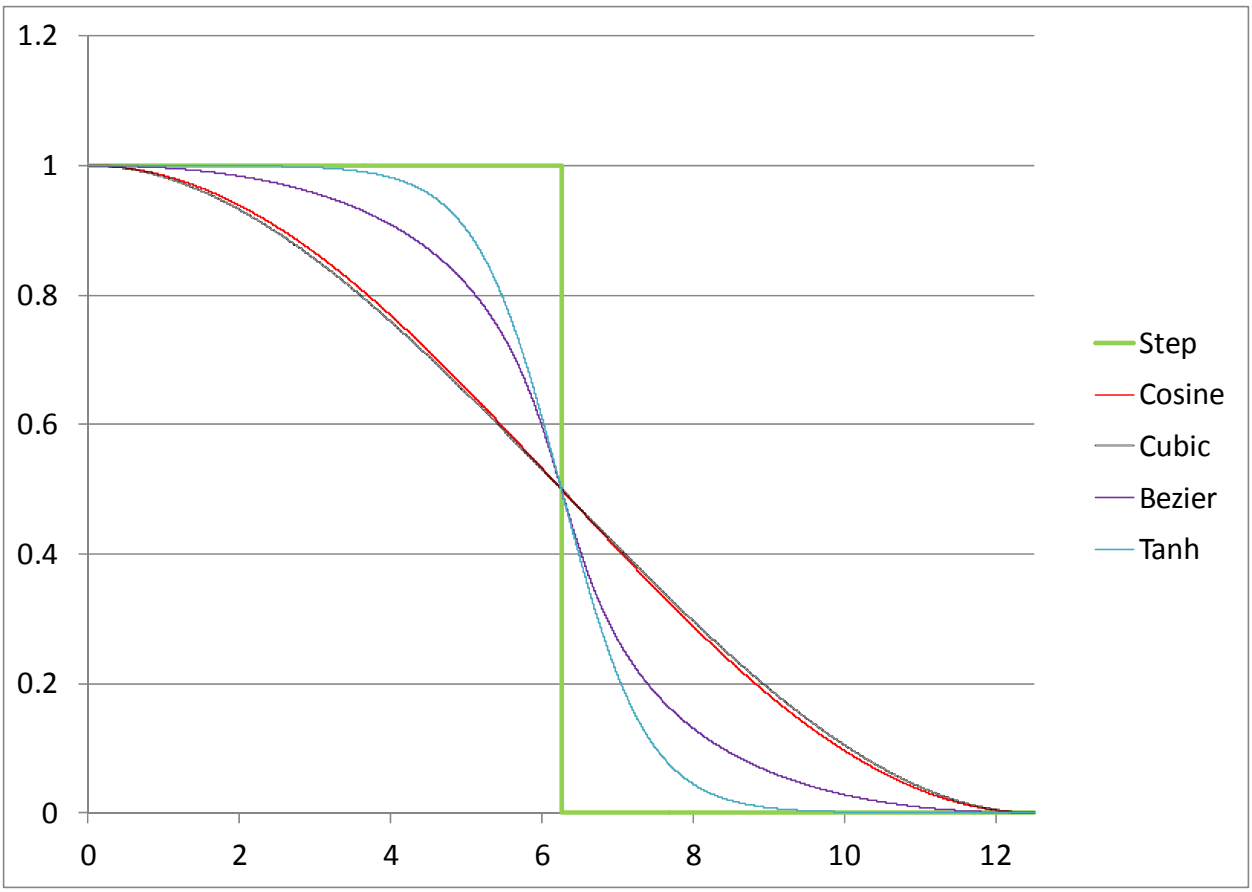


Figure 6 – Zoomed Figure 5

Table 1 shows the relative difference between the smooth step functions for the two cases of epsilon.

Table 1 – e3 Error

Epsilon, % of tau	Error (% of pulse area)			
	Cosine	Cubic	Bezier	Tanh
10	7.2676	7.5000	4.0000	2.5205
50	36.3380	37.5000	20.000	12.6024

The values in Table 1 correlate with Figure 6 and suggest the "best" smooth step approximation rank order to be tanh, Bezier, cosine, and cubic in terms of e3 error.

5.0 COSINE

The cosine function is plotted in Figure 7 for a domain of 0 to 2π (1 period). Looking at the first half of the interval there is a smooth transition between 1 and -1. With translation and scaling the function can be modified to make a smooth step between defined levels as in Figure 1.

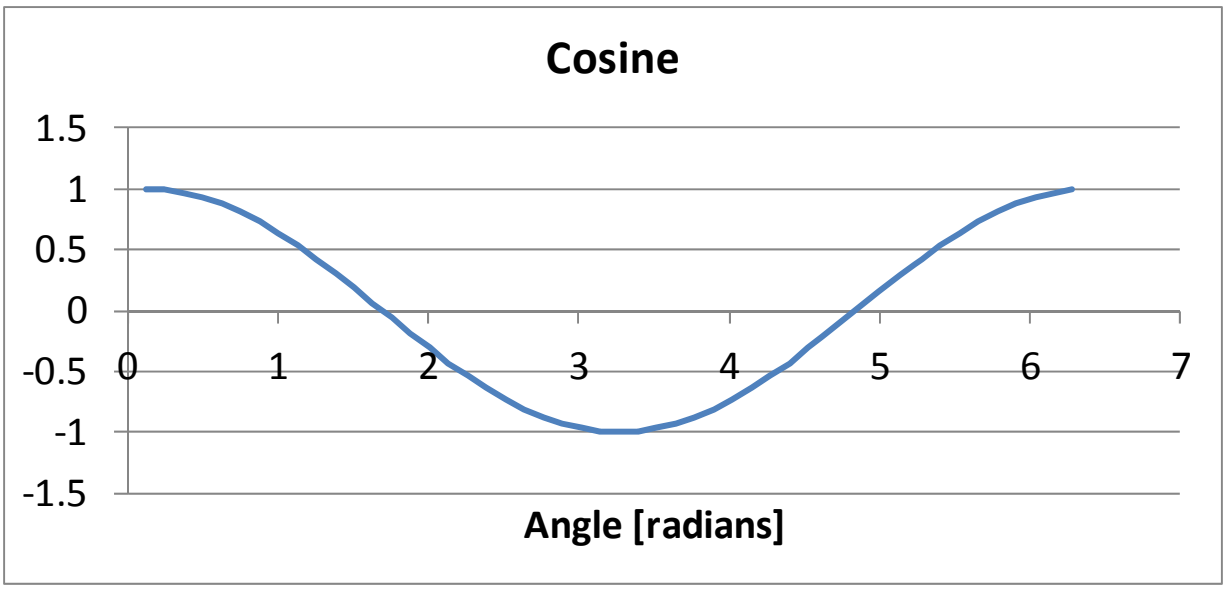


Figure 7 – Cosine Function

Each side of the center pulse in Figure 2 will use the first half (right side) or second half (left side) of the cosine period. Scaling (or normalizing) the amplitude to obtain an output value between zero and one is accomplished by adding 1 and multiplying by 0.5 as

$$\frac{1}{2}(1+\cos nw_0 t) \tag{13}$$

Translation is used to transform the time scale for the square pulse into the desired half of the cosine function time scale. With t_c being the cosine function time scale, for the right side of the center square pulse the desired shift is

$$\begin{gathered} 0 < t_c < T/2 \\ 0 < t_c < 2\varepsilon \\ 0 < t - \left(\frac{\tau}{2} - \varepsilon\right) < 2\varepsilon \end{gathered} \tag{14}$$

while for the left side it is

$$\begin{gathered} T/2 < t_c < T \\ 2\varepsilon < t_c < 4\varepsilon \\ 2\varepsilon < t - \left(-\frac{\tau}{2} - 3\varepsilon\right) < 4\varepsilon \end{gathered} \tag{15}$$

A more straight forward approach is to use the shift in (14) for both sides but change the sign after the one in (13).

Implementation specifics are denoted using MATLAB notation. From (14)

$$\frac{T}{2} = 2\varepsilon \;\rightarrow T = 4\varepsilon \tag{16}$$

the cosine period is

```
% set period
   tc = 4*epi;
```

For the specific example in this report with the following parameters

```
%=== parameters
   T = 25;                          % period length
   tau = 12.5;                      % square pulse width
   d = tau/T;                       % duty ratio
   dt = 0.001;                      % time step
   t = [-50:dt:50];                 % time vector
   iierr = find( abs(t) < T/2 );

%=== loop for each epsilon
   for epi =[ 0.01:.01:.1,.5]*tau, % percentage of tau
```

the cosine step function is defined as

```
   ii = find( abs(t + tau/2) <= epi );
   stepm(ii) = 0.5 * (1 - cos( (2*pi/tc)*(t(ii) - (-tau/2-epi)) ) );
   ii = find( abs(t - tau/2) <= epi );
   stepm(ii) = 0.5 * (1 + cos( (2*pi/tc)*(t(ii) - (tau/2-epi)) ) );
```

Calculation of the Fourier coefficients is based on (9) and (10)

```
   trange = find ( (t >= -tau/2-epi) & (t <= tau/2+epi) );

   coef = trapz( stepm(trange).*exp(-j*2*pi*n.*t(trange)/T) );
   coef=1/T*real(coef)*dt;

   % handle constant term separate
   if (n == 0),
      y(cnt,:) = coef + 0*t;
   else
      y(cnt,:) = y(cnt-1,:) + 2 * coef*cos(n*2*pi/T.*t);
   end
   cnt = cnt + 1;

   % compute discrete step coefficients directly
   if ( n == 0 ),
      yo(cnto,:) = d * sinc(n*d) * cos(n*2*pi/T.*t);
   else
      yo(cnto,:) = yo(cnto-1,:) + 2 * d*sinc(n*d)*cos(n*2*pi/T.*t);
   end
   cnto = cnto + 1;
```

The error computation follows from (11)

```
   % continuous step
   % cosine
   err2 = trapz( abs( y(nnum+1,iierr) - stepm(iierr) ) ) * dt;

   % difference
   % cosine
   err3 = trapz( abs( stepm(iierr) - step(iierr) ) ) * dt;
```

Figure 8 shows the cosine function partial Fourier series sum error results for e1 (red, center), e2 (green, lower left), and e2+e3 (blue, upper right) for epsilon as a percentage of tau. Note that e1 is a function of N only, and e3 is a function of epsilon only. As epsilon increases, the e2 error is reduced quickly for a small number of terms. For e2+e3, as epsilon increases, e3 begins to dominate and reduces the number of terms at which the condition in (12) is satisfied. The “squiggles” in the curves are attributed to the odd coefficients being zero which results in a constant sum for two consecutive terms. This principle is described in [4] for the normalized sinc function.

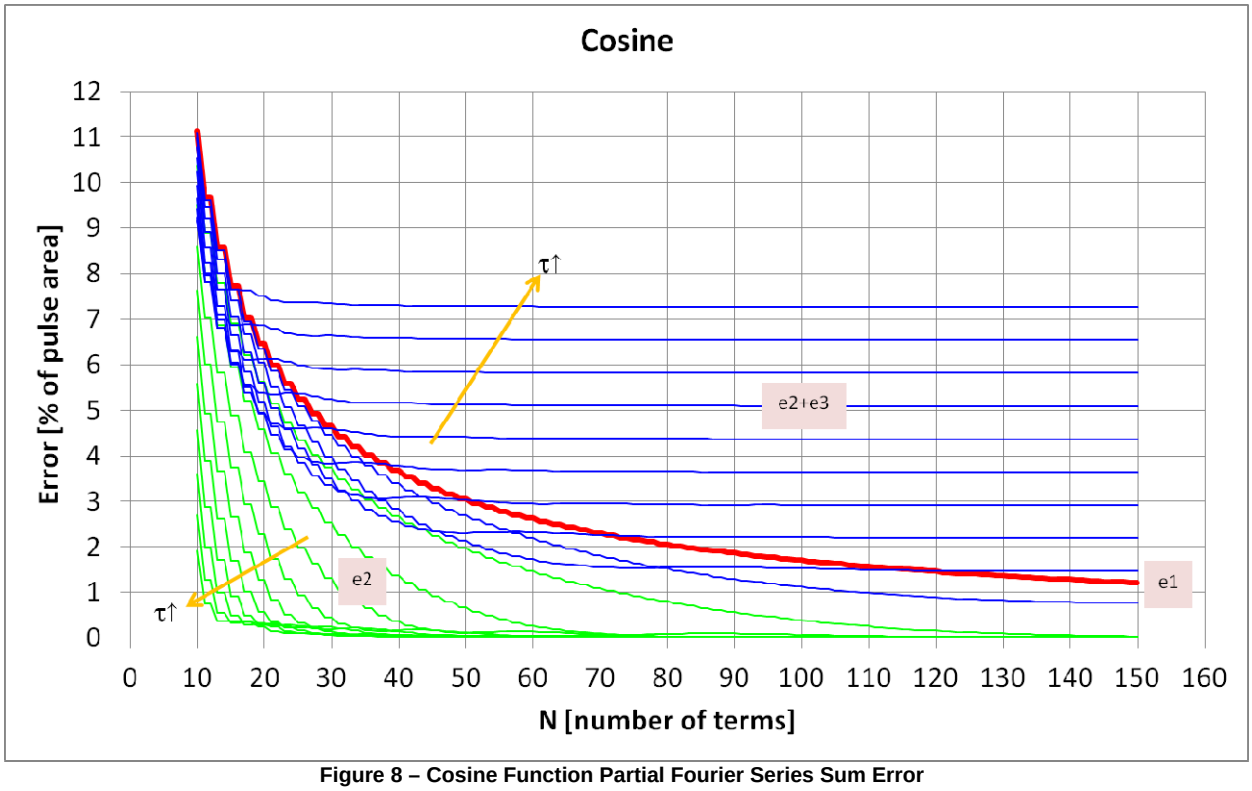


Figure 8 – Cosine Function Partial Fourier Series Sum Error

6.0 CUBIC POLYNOMIAL

The cubic polynomial function

$$y = ax^3 + bx^2 + cx + d \tag{17}$$

is plotted in Figure 9 for a domain of 0 to 1 that provides a smooth transition between 0 and 1 assuming the derivative at each end point is equal to zero. With these conditions

$$y(0) = 0, y(1) = 1, y'(0) = 0, y'(1) = 0.$$

we arrive at

$$y = -2x^3 + 3x^2 = xx(3 - 2x). \tag{18}$$

As described in [5] and [6], the domain is normalized using a “clamping” function defined below

```
% clamp
function y = clamp(x,min,max)

y = x;

% limit x values to min/max
imax = find(y > max);
y(imax) = max;
imin = find(y < min);
y(imin) = min;
```

which is used in defining the cubic polynomial or “smooth step” as

```
% smoothstep
function y = smoothstep(x,min,max)

% normalize (0 to 1) using min/max in domain of x
xx = (x-min)/(max-min);
s_min = 0.0;
s_max = 1.0;

% set domain limits
y = clamp(xx, s_min, s_max);

% return value
y = y .* y .* (3 - 2*y);
```

The left side of the center pulse in Figure 2 will use the normalized domain between +/- epsilon and the right side will use a reversed normalized domain (or subtract the left side result from one) with appropriate amplitude scaling (in this case just one).

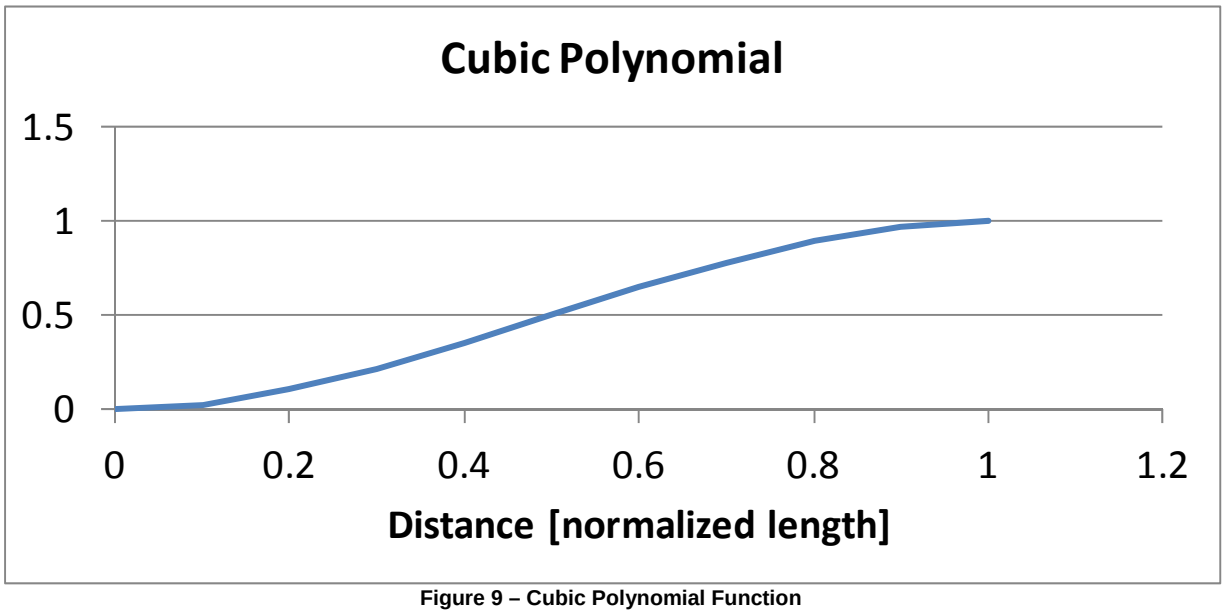


Figure 9 – Cubic Polynomial Function

With the same parameters and definitions as outlined in Section 5.0, with exceptions called out, the cubic polynomial step function is defined as

```
% cubic
   ii = find( abs(t+tau/2) <= epi );
   stepm(ii) = smoothstep( t(ii), min(t(ii)), max(t(ii)) );
   ii = find( abs(t-tau/2) <= epi ) ;
   iii = t(ii);
   rt = iii(end:-1:1);
   stepm(ii) = smoothstep( rt, min(rt), max(rt) );
%   stepm(ii) = 1 - smoothstep( t(ii), min(t(ii)), max(t(ii)) );
```

Figure 10 shows the cubic polynomial partial Fourier series sum error results. Similar results are obtained as in Section 5.0.

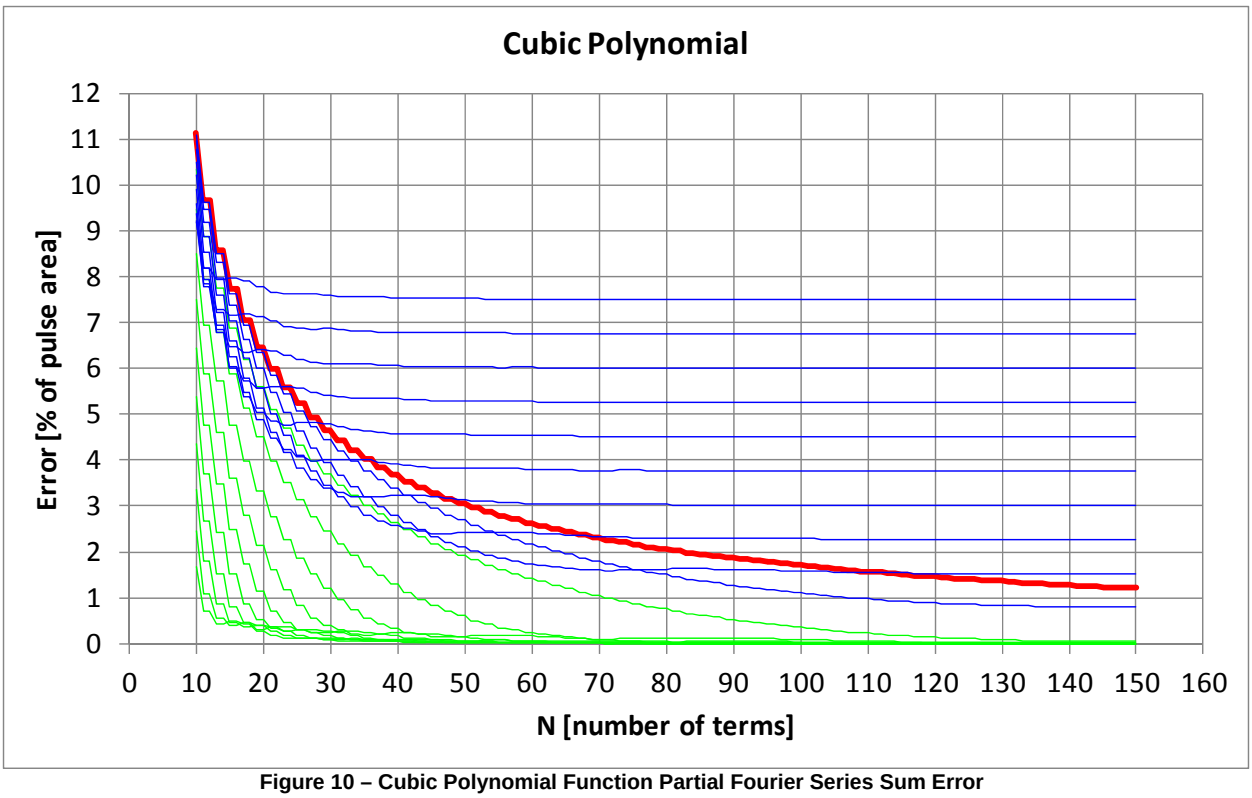


Figure 10 – Cubic Polynomial Function Partial Fourier Series Sum Error

7.0 BEZIER POLYNOMIAL

The Bezier polynomial function is similar to the cubic polynomial in Section 6.0 except that both x and y values are defined as a function of parameter t with a domain of 0 to 1

$$x = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y = a_y t^3 + b_y t^2 + c_y t + d_y \tag{19}$$

Four points, P0, P1, P2, and P3 of (x,y), are needed to solve for each variable as discussed in [7]. The procedure is the same as for the cubic polynomial in Section 6.0 except that the slopes at the end points are defined in terms of the parameter t as

$$x(0) = d_x = P0x$$

$$x(1) = a_x + b_x + c_x + d_x = P3x$$

$$x'(0) = \ c_x = slope = \frac{rise}{run} = \frac{P1x - P0x}{\frac{1}{3} - \ 0} = 3(P1x - P0x)$$

$$x'(1) = 3a_x + 2b_x + c_x = \frac{P3x - P2x}{1 - \frac{2}{3}} = 3(P3x - P2x) \tag{20}$$

.

The same steps are followed to determine y. In matrix form, the coefficients defined in terms of the points are

$$\begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P0x \\ P1x \\ P2x \\ P3x \end{bmatrix} \tag{21}$$

With the points

$$P0 = (0,0), P1 = (0.8, 0), P2 = (0.2,1.0), P3 = (1.0,1.0) \tag{22}$$

the coefficients for x and y are (2.8, -4.2, 2.4,0) and (-2, 3, 0, 0), respectively. Figure 11 shows the difference between the Bezier and cubic polynomials. Adjustments to the point

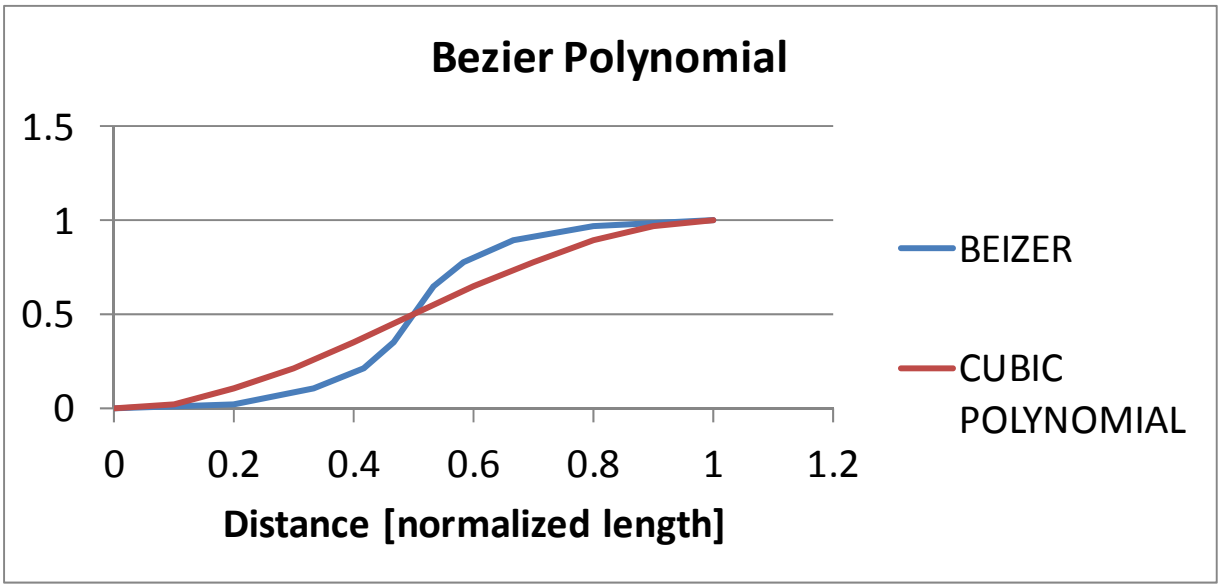


Figure 11 – Bezier Polynomial Function

definitions allow for manipulation of the curve shape. For example, redefining the points in (22) to

$$P1 = (0.33, 0), P2 = (0.66,1.0)$$

the Bezier becomes the cubic polynomial with x coefficients of (0.01, 0, 0.99, 0). Changes to the "smooth step" function in Section 6.0, to allow for more flexibility with the Bezier polynomial, are included below.

```
% smoothstepbez
function y = smoothstepbez(x,min,max,ind)

if (ind == 'x')
   p=[0 0.8 0.2 1]';
elseif (ind == 'y')
   p=[0 0 1 1]';
else
   p=[0 .33 .66 1]';  % cubic
end
```

```
T = [-1 3  -3 1;
   3  -6 3  0;
   -3 3  0  0;
   1  0  0  0];
c = T*p;

% normalize (0 to 1) using min/max in domain of x
xx = (x-min)/(max-min);
s_min = 0.0;
s_max = 1.0;

% set domain limits
y = clamp(xx,s_min,s_max);

% return value
y = c(1)*y.^3 + c(2)*y.^2 + c(3)*y + c(4);
```

As for the cubic polynomial in Section 6.0, the left side of the center pulse in Figure 2 will use the normalized domain between +/- epsilon and the right side will use a reversed normalized domain (or subtract the left side result from one) with appropriate amplitude scaling.

With the same parameters and definitions as outlined in Section 5.0, with exceptions called out, the Bezier polynomial step function is defined as

```
% Bezier
   xt = t;
   ii = find( abs(t + tau/2) <= epi );
   xt_start = t(ii(1));
   xt_len = t(ii(end)) - t(ii(1));

   % define x domain
   xt_i = xt_start + xt_len * smoothstepbez( t(ii), min(t(ii)), max(t(ii)), 'x' );
   xt = [xt(1:ii(1)-1), xt_i, xt(ii(end)+1:end)];

   % interpolate t, using x, to get y
   stepm(ii) = interp1(xt(ii), smoothstepbez( t(ii), min(t(ii)), max(t(ii)), 'y'), t(ii),
        'linear', 'extrap');

   ii = find( abs(t - tau/2) <= epi );
   iii = t(ii);
   rt = iii(end:-1:1);
   xrt_start = rt(1);
   xrt_len = rt(end) - rt(1);

   % define reverse x domain
   xrt = xrt_start + xrt_len * smoothstepbez( rt, min(rt), max(rt), 'x' );
   xt = [xt(1:ii(1)-1), xrt, xt(ii(end)+1:end)];

   % interpolate t, using reverse x, to get y
   stepm(ii) = interp1( xt(ii), smoothstepbez( rt, min(rt), max(rt), 'y'), t(ii), 'linear',
        'extrap' );
%   stepm(ii) = 1 - interp1(xt(ii), smoothstepbez( t(ii), min(t(ii)), max(t(ii)), 'y'), t(ii),
       'linear', 'extrap');
```

In this case, because x is also defined as a function of the independent parameter t, the interpolation function is used to obtain the associated x value for the time grid which is used as the independent variable for the other step functions.

Figure 12 shows the Bezier polynomial partial Fourier series sum error results. Differences are seen between the cosine and cubic functions in that while the e3 error is smaller, more terms are needed to reduce the e2 error for larger values of epsilon as a percentage of tau. However, all the e2 errors are less than e1.

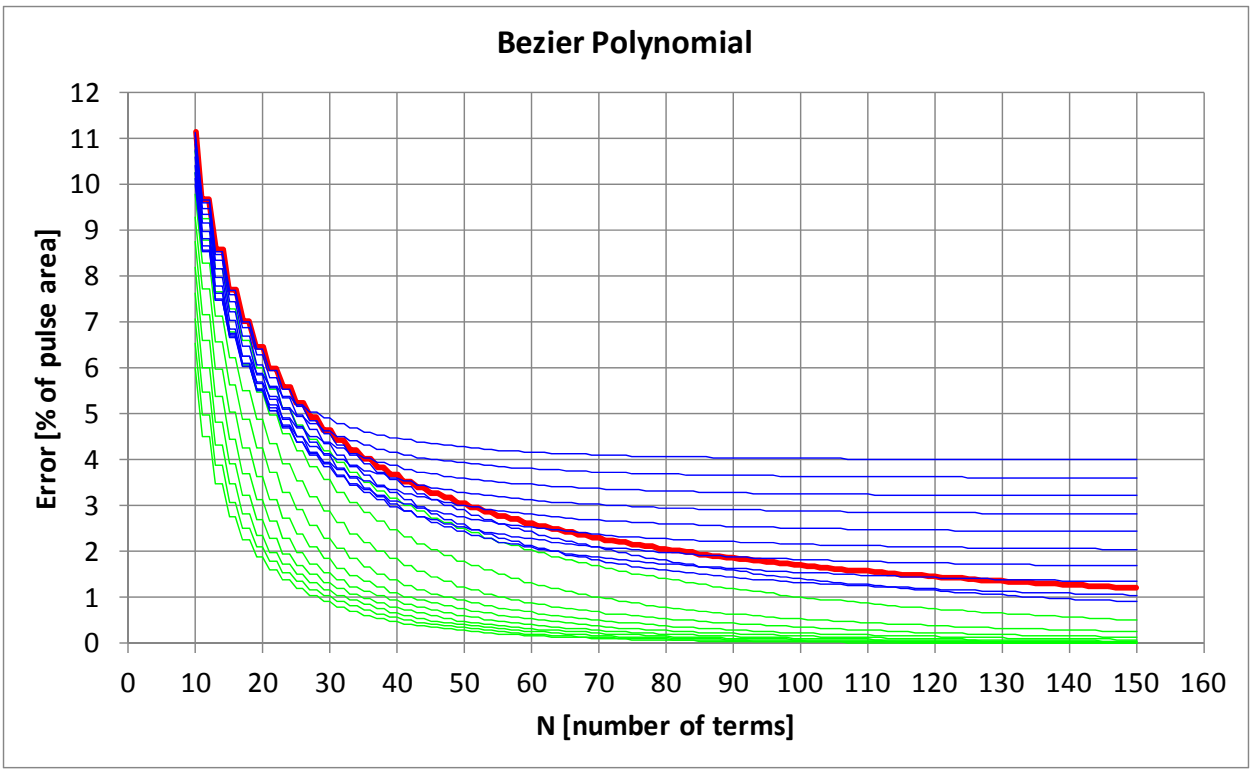


Figure 12 – Bezier Polynomial Function Partial Fourier Series Sum Error

8.0 HYPERBOLIC TANGENT

The hyperbolic tangent (tanh) function is plotted in Figure 13. Looking at the center there is a smooth transition between 1 and -1---although the function approaches 1 (-1) as the domain approaches infinity (-infinity). As with the cosine function in Section 5.0, with translation and scaling the tanh function can be modified to make a smooth step between defined levels as in Figure 1.

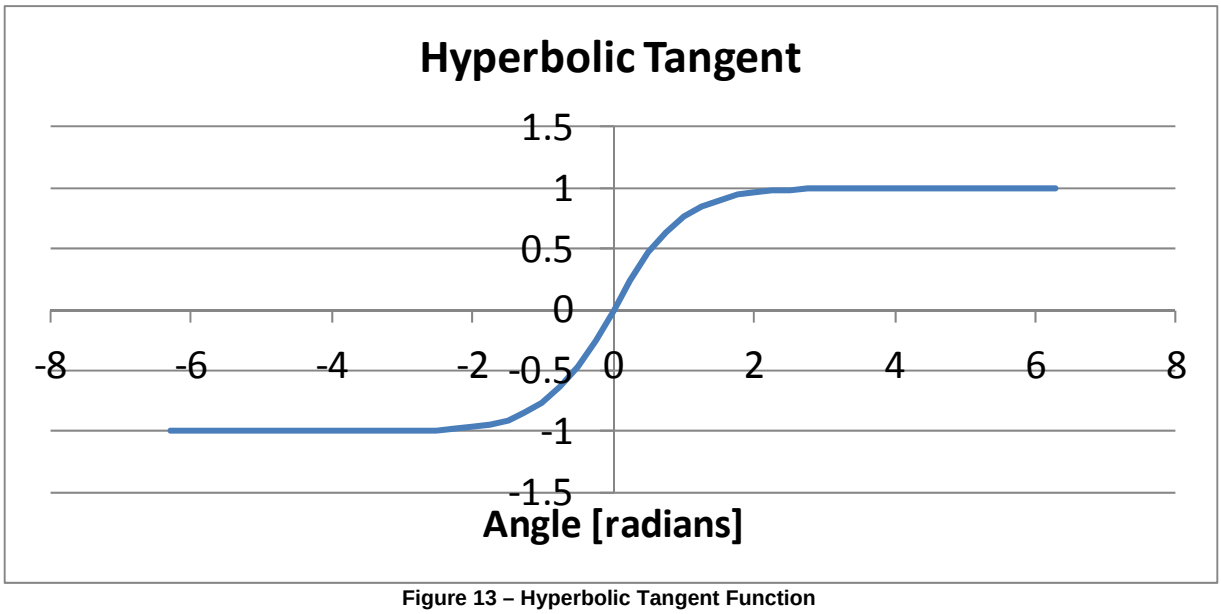


Figure 13 – Hyperbolic Tangent Function

An output value between zero and one is accomplished by adding 1 and multiplying by 0.5 as

$$\frac{1}{2}(1 + \tanh\ bx) \tag{23}$$

The sign after the one in (23) is used to change the direction of the level transition.

Due to machine precision the tanh function will achieve the value of 1 in some interval, even though it approaches infinity in theory. Matching the transition space in Figure 1 to that of the tanh function requires a scaling factor, b, which can be found from the test case below

```
% b.m
t = [-10:.1:10];                % time scale
y = find( abs( round( tanh(t)*10000 )/10000) >= 1.0 );
min( abs(t(y)) )
```

which gives 5.3. With the ranges

$$-\varepsilon < x < \epsilon$$
$$-5.3 < bx < 5.3$$

then

$$b = \frac{5.5}{\varepsilon}$$

where the value of 5.3 was increased to provide some margin of safety.

With the same parameters and definitions as outlined in Section 5.0, with exceptions called out, the hyperbolic tangent step function with the scaling factor

```
   % tanh
   b = 5.5/epi;
```

is defined as

```
% tanh
   ii = find( abs(t + tau/2) <= epi );
   stepm(ii) = 0.5 * (1 + tanh( b*(t(ii) - (-tau/2)) ));
   ii =find( abs(t - tau/2) <= epi ) ;
   stepm(ii) = 0.5 * (1 - tanh( b*(t(ii) - (tau/2)) ));
```

Figure 14 shows the hyperbolic tangent partial Fourier series sum error results. The e3 error is smaller and the e2+e3 curves are converging to the e1 error as tanh more closely approximates the discrete step as indicated by Table 1 under the condition in (12). As compared to the cosine and cubic functions, more terms are required to reduce the e2 error.

As a side note, another use for the tanh function, which is not covered in detail here, is for blending functions to provide a smooth transition between them as described in [8]. With s(x) defined in (23) the result is

$$h(x) = s(x)f_1(x) + \big(1 - s(x)\big)f_2(x).$$

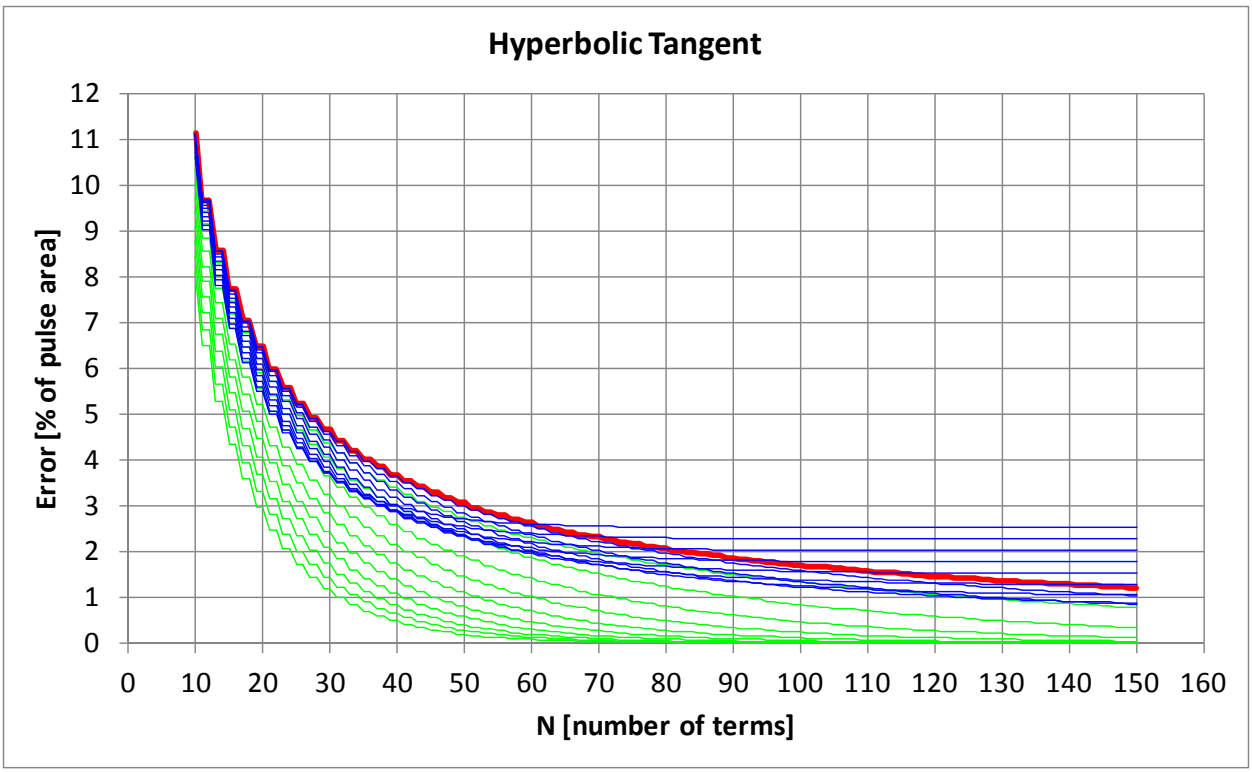


Figure 14 – Hyperbolic Tangent Function Partial Fourier Series Sum Error

9.0 RESULTS AND SUMMARY

The partial Fourier series sums error results (Figures 8, 10, 12, and 14) for the cosine, cubic polynomial, Bezier polynomial, and hyperbolic tangent step functions, respectively, demonstrate that the smoother the function (in terms of e3, larger e3 indicates smoother) the more quickly the error is reduced for a smaller number of Fourier series terms. Computation of the coefficient terms is outlined in Appendix A.2. The error data was processed to determine the maximum number of terms at which condition (12) was met. This procedure is outlined in Appendix A.3. The summarized data is presented in Figure 15. Note that since the amplitude of the square pulse in Figure 2 is one, the percentage of tau is equivalent to the percentage of the pulse area. As epsilon increases the size of e3 does also and is independent of the number of terms N (constant as a function of N). This pushes the intersection point (between the e2+e3 and e1) to a lower (leftward) number of terms (N) as Figures 10 and 14 demonstrate. The continuous step functions that have the largest e3 error (see Table 1) are restricted to the lowest number of partial sum terms, although reducing the error more quickly.

Figures 16, 17, and 18 show the e1, e2, and e3 errors, respectively, at the error limit condition in Figure 15 (N is not shown on the plots). Details for each function are included in Table 2.

The e1 error, in Figure 16, is greater than 1% in all cases and follows the same order as in Figure 15 where for fewer number of terms the error is larger (epsilon increases => e3 increases => N decreases => e1 increases).

The e2 error, in Figure 17, is less than 1% for all epsilon except for the Bezier Polynomial (which is greater than 1% for epsilon greater than 8%), but still less than the associated e1 error. This indicates that at the error limit condition the continuous step functions have smaller approximation error than the discrete step partial Fourier series sums.

The order change in the e2 error of Figure 17 is explained in Figures 18 and 19 by the cross-over of the Bezier and tanh functions for some intermediate value of epsilon between 1% and 10%. At this value of epsilon, the e2 error decreases quicker than the Bezier function for larger values of N. This illustrates one advantage of the tanh function for an epsilon greater than 4%.

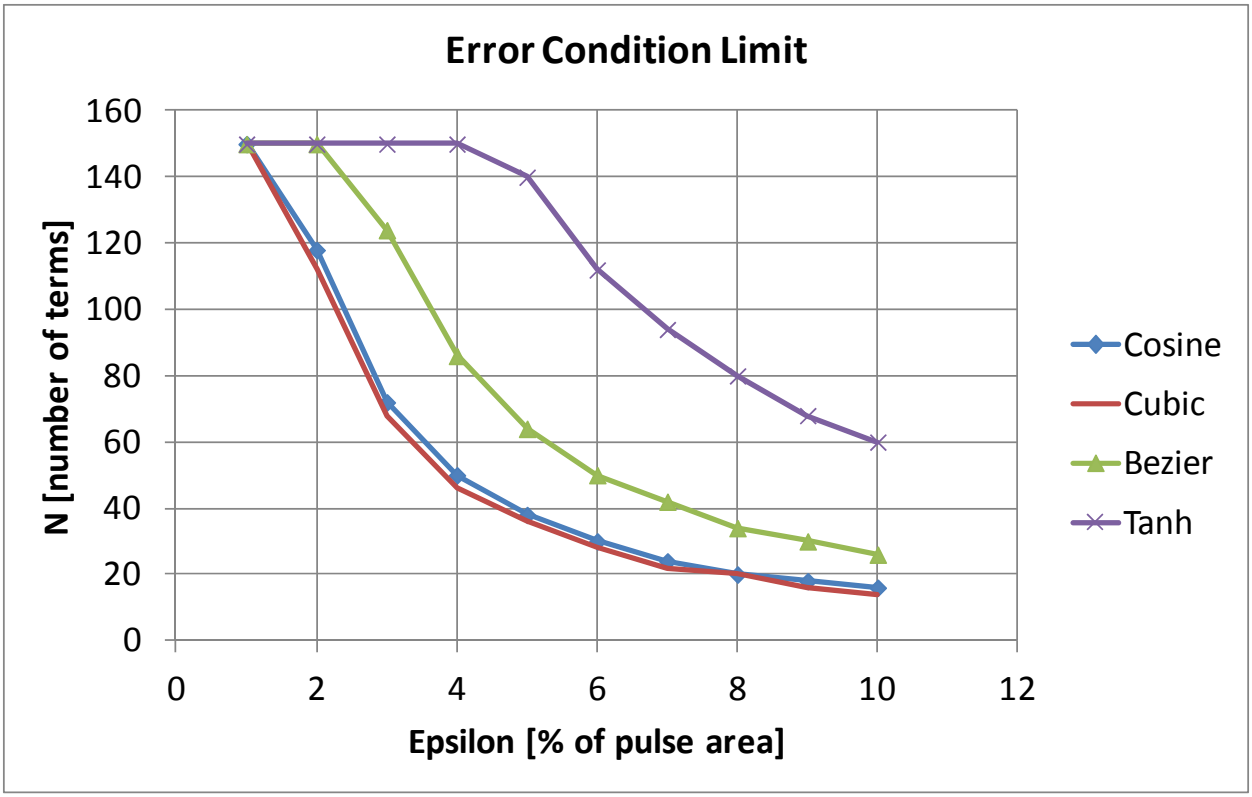


Figure 15 –Partial Fourier Series Sums Error Results

The e3 error, in Figure 18, has constant slope consistent with Table 1 as it is independent of N.

The e2+e3 error in Figure 21 shows that the e3 error order of continuous step functions is preserved, except for small epsilon (see Figure 17).

From this analysis and the error criteria in (12), we can conclude that the overall "best" continuous step function approximation to a discrete step is the Hyperbolic Tangent (tanh) function for epsilon greater than approximately 1.5% and the cosine function for epsilon less than approximately 1.25%.

Future work in this area would be to analyze the errors for epsilon less than 1% and the transition area between epsilon of 1% to 1.5%.

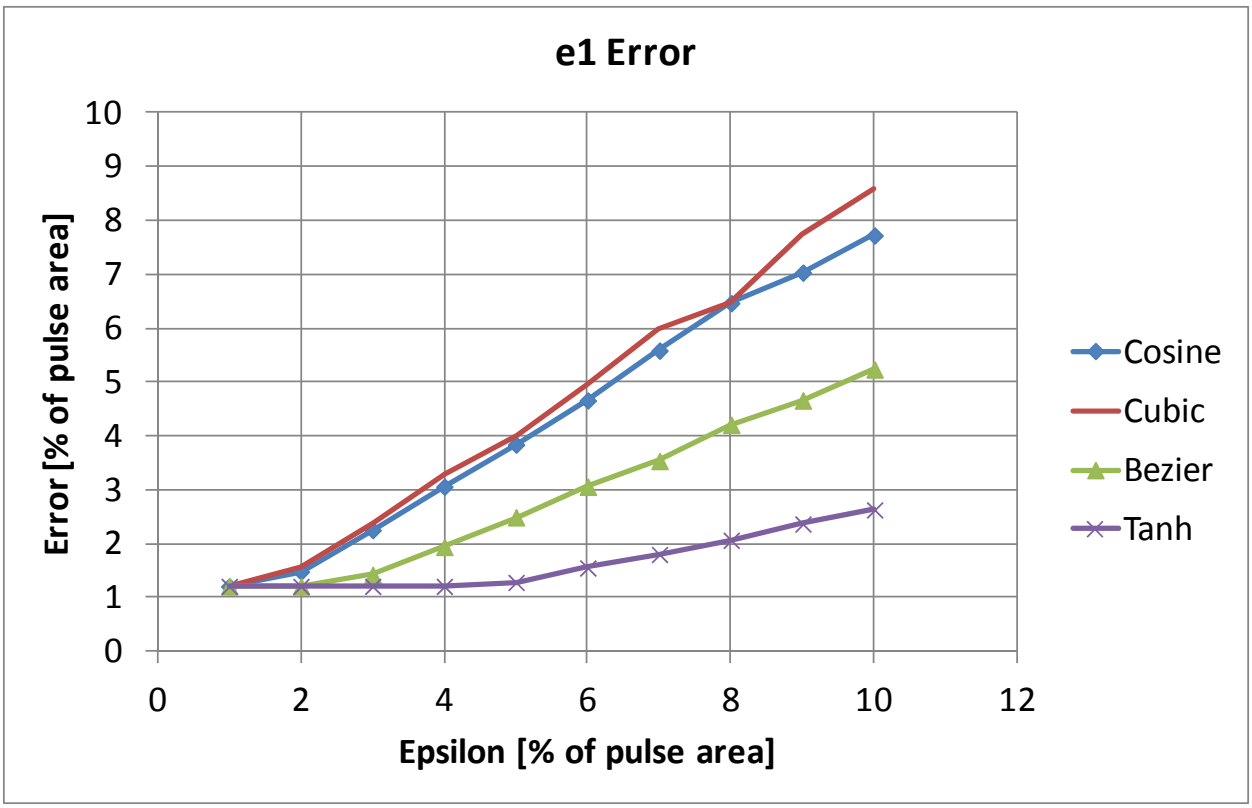


Figure 16 –e1 Error (for N in Fig. 15)

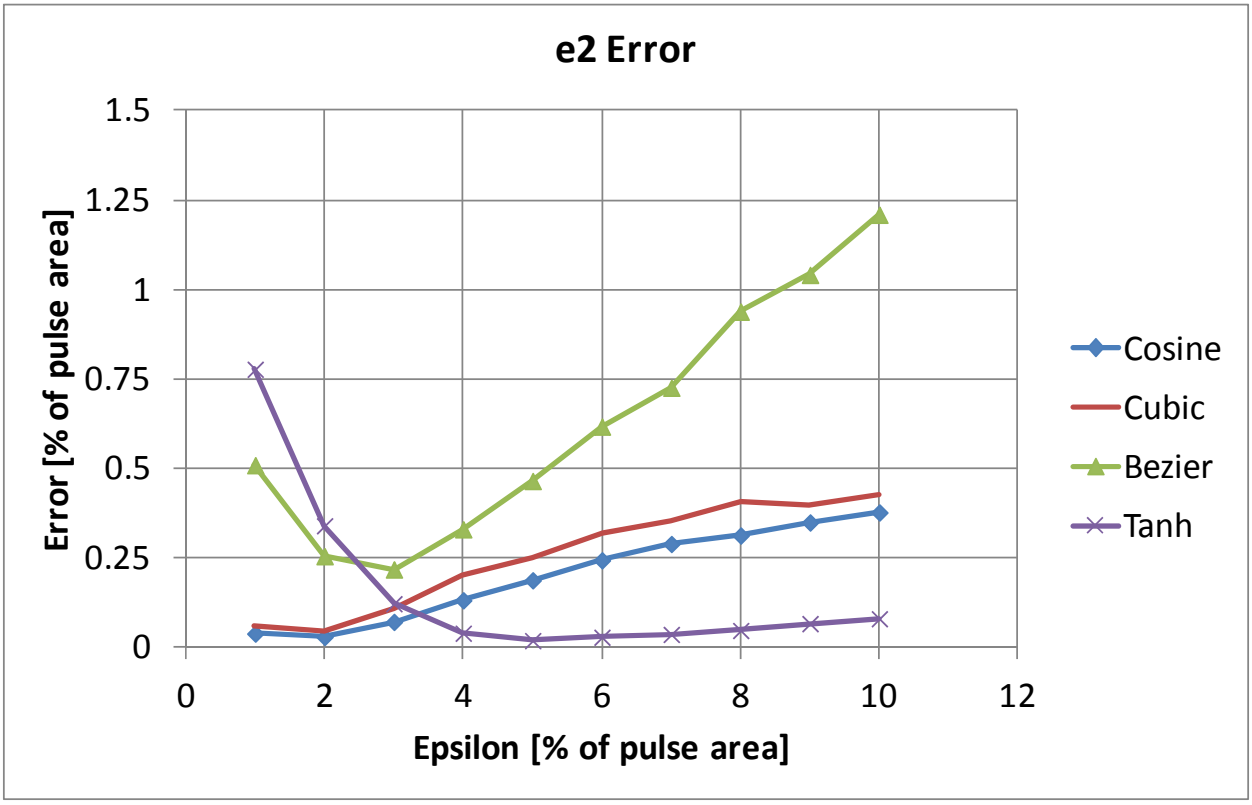


Figure 17 –e2 Error (for N in Fig. 15)

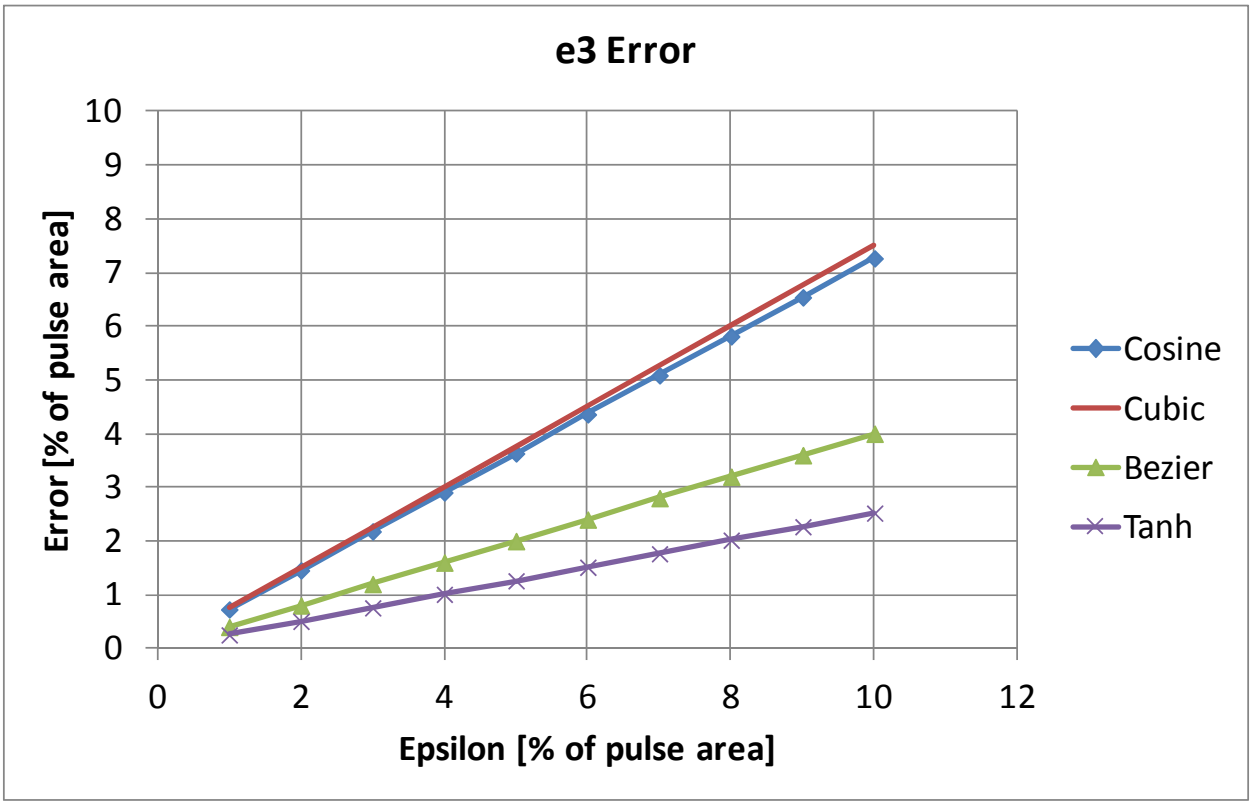


Figure 18 –e3 Error (for N in Fig. 15)

Table 2 – Maximum N Meeting Error Condition (12) and e1, e2, e3 Values

Function	epsilon	N	e1	e2	e3
Cosine	1	150	1.209537	0.039066	0.726777
	2	118	1.485054	0.029608	1.453529
	3	72	2.256793	0.071739	2.180287
	4	50	3.061496	0.132045	2.907046
	5	38	3.841763	0.187851	3.633806
	6	30	4.662628	0.243676	4.360566
	7	24	5.587874	0.289478	5.087326
	8	20	6.469539	0.312949	5.814086
	9	18	7.036793	0.349583	6.540846
	10	16	7.725602	0.377597	7.267606
Cubic	1	150	1.209537	0.059059	0.750016
	2	112	1.552625	0.0463	1.500008
	3	68	2.367846	0.107298	2.250005
	4	46	3.280907	0.201364	3.000004
	5	36	4.016389	0.251514	3.750003
	6	28	4.931978	0.319425	4.500003
	7	22	5.993539	0.353315	5.250002
	8	20	6.469539	0.405416	6.000002
	9	16	7.725602	0.396347	6.750002
	10	14	8.581817	0.426217	7.500002
Bezier	1	150	1.209537	0.509525	0.400092
	2	150	1.209537	0.255185	0.800046
	3	124	1.423539	0.218305	1.20003
	4	86	1.942723	0.330454	1.600023
	5	64	2.491377	0.465372	2.000018
	6	50	3.061496	0.617477	2.400015
	7	42	3.537433	0.72593	2.800013
	8	34	4.209219	0.938606	3.200012
	9	30	4.662628	1.041265	3.60001
	10	26	5.237663	1.209506	4.000009
Tanh	1	150	1.209537	0.777596	0.252106
	2	150	1.209537	0.340412	0.504124
	3	150	1.209537	0.122548	0.756162
	4	150	1.209537	0.040091	1.008205
	5	140	1.283229	0.019434	1.260249
	6	112	1.552625	0.028767	1.512295
	7	94	1.801826	0.036689	1.764341
	8	80	2.065102	0.047711	2.016387
	9	68	2.367846	0.066982	2.268434
	10	60	2.629701	0.080618	2.520481

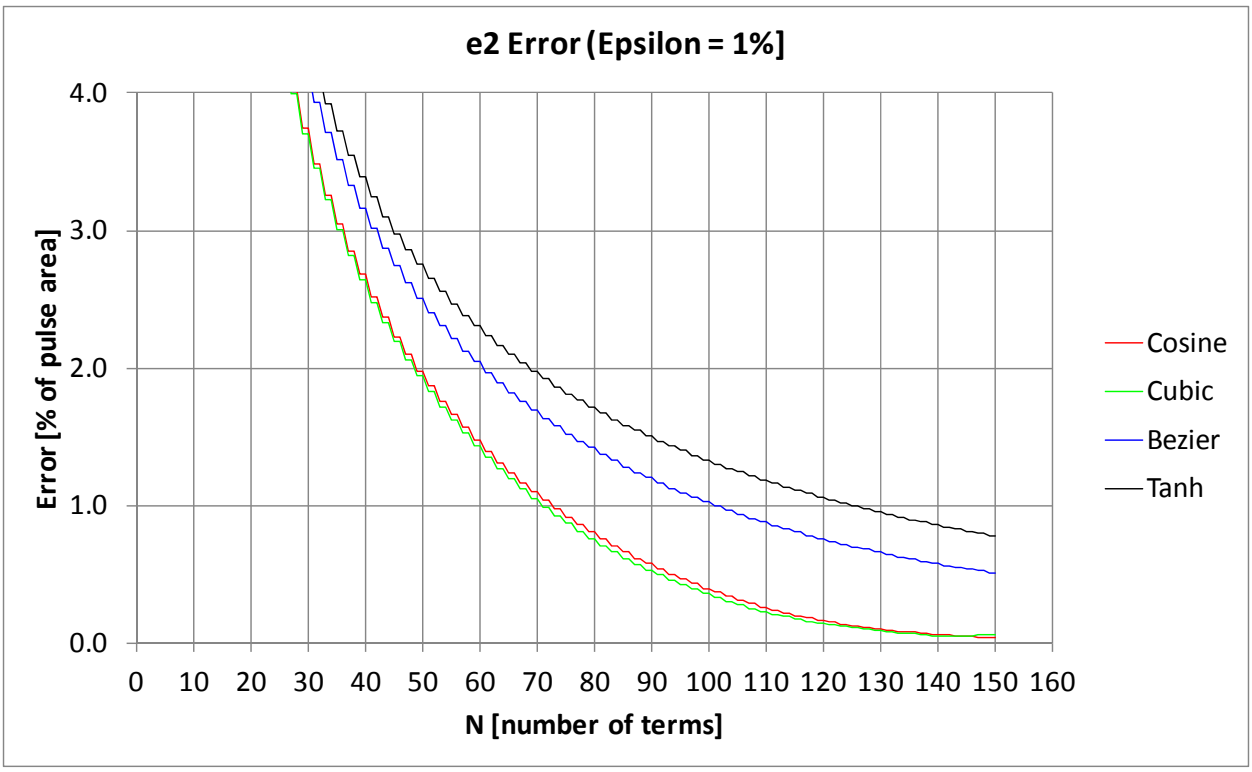


Figure 19 –e2 Error (for N in Fig. 15, epsilon = 1%)

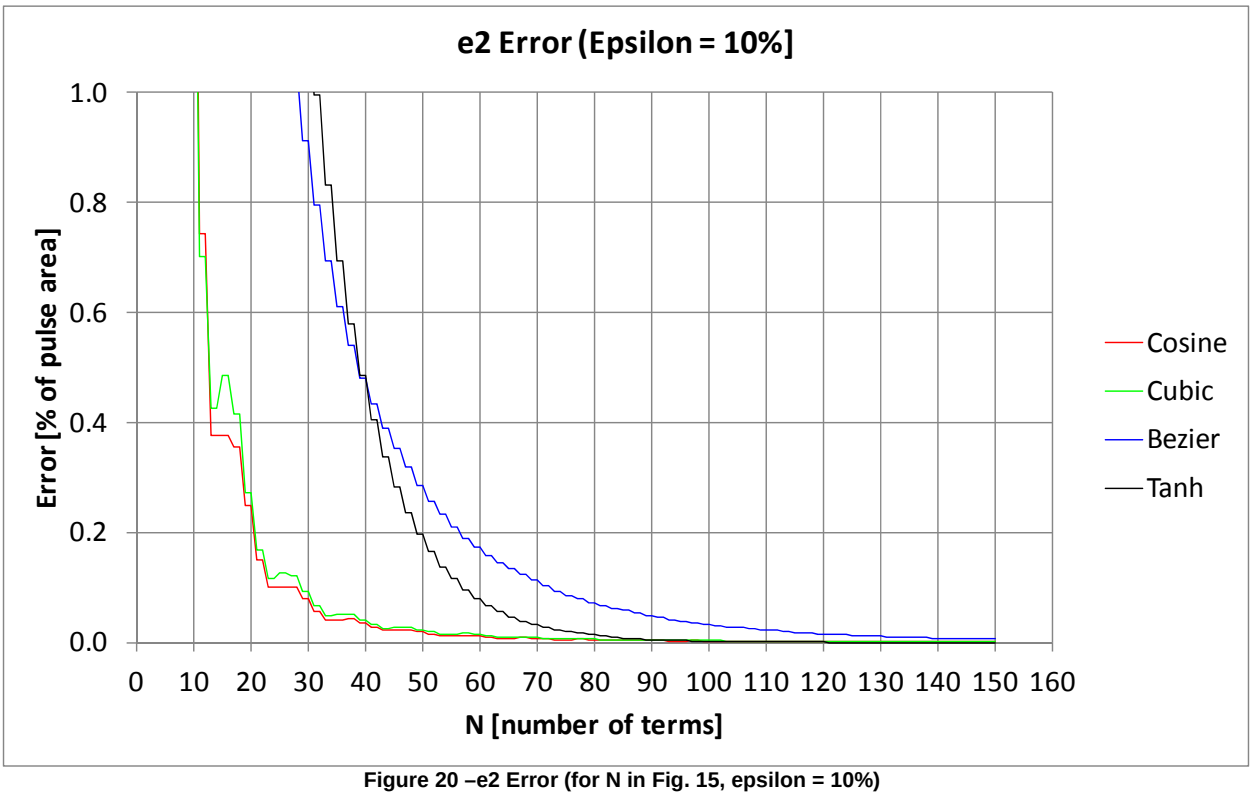


Figure 20 –e2 Error (for N in Fig. 15, epsilon = 10%)

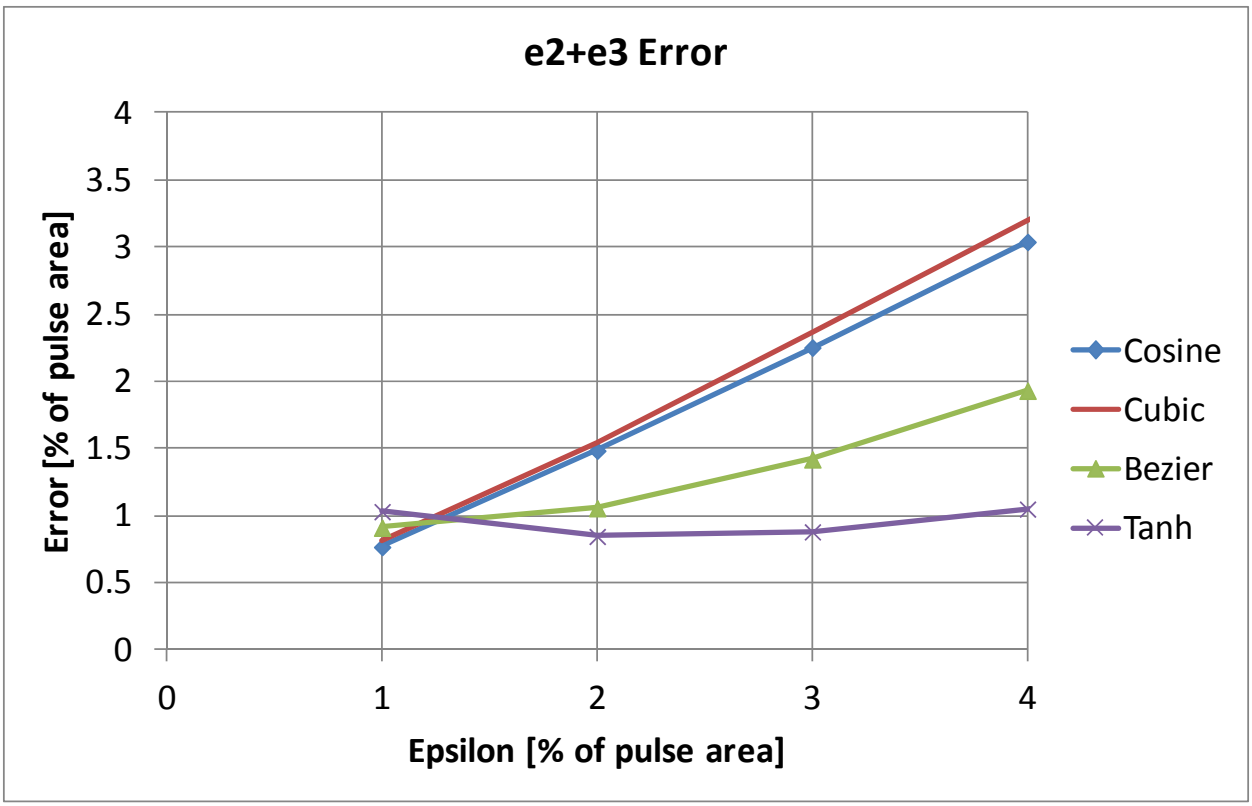


Figure 21 –e2+e3 Error (for N in Fig. 15)

REFERENCES

[1] ADAMS/Solver (STEP function) (http://www.mscsoftware.com).

[2] Stark, et.al, Modern Electrical Communications, 2nd. Edition. Prentice Hall, NJ. ISBN 0-13-593278-5.

[3] MATrix LABoratory (MATLAB) (http://www.mathworks.com).

[4] Sinc Function (http://en.wikipedia.org/wiki/Sinc_function).

[5] Smooth Step (http://en.wikipedia.org/wiki/Smoothstep).

[6] Clamping (http://en.wikipedia.org/wiki/Clamping).

[7] Bezier (http://www.cs.binghamton.edu/~reckert/460/bezier.htm).

[8] Tanh Smoothing (http://www.j-raedler.de/2010/10/smooth-transition-between-functions-with-tanh/).

APPENDIX A.1 – INTEGRATION DIFFERENCES

```
%intdiff.m

clear all
close all

%=== parameters
T = 5;            % period length
tau = 1;          % square pulse width
d = tau/T;        % duty ratio
err = [];         % error results
nnuml = 50;       % number of terms

%=== loop for time step
for dt = [1e-1 1e-2 1e-3 1e-4 1e-5 1e-6 1e-7 1e-8 1e-9 1e-10],

   t = [-tau/2:dt:tau/2];
   med = eps;  % max error (direct)
   mei = eps;  % max error (numerical)

   % loop coefficients
   for n = 1:nnuml,

      % closed form
      cn = 2 * d*sinc(n*d);

      % direct
      di = 2 * (( exp(-j*2*pi*n*(tau/2)/T)/(-j*2*pi*n) ) –
            ( exp(-j*2*pi*n*(-tau/2)/T)/(-j*2*pi*n) ));

      % numerical
      ni = trapz( exp(-j*2*pi*n.*t/T) );
      ni = 2 * 1/T*real(ni)*dt;

      % maximum error
      med = max( abs(cn-di), med );
      mei = max( abs(cn-ni), mei );

      % output 'term #, closed form, direct(di), err w/di, numerical(ni), error w/ni'
      fprintf('%10.8f %3d %20.14f %20.14f %20.14f %20.14f %20.14f\n',dt,n,cn,di ,cn-di,ni,cn-ni);

   end
   fprintf('\n');
   fprintf('max err:  %20.14f %20.14f\n',-log10(med),-log10(mei));
   fprintf('\n');

   % results
   err = [err;[-log10(dt),-log10(med),-log10(mei)]];

end

% output results
fid = fopen('intout.txt','wt');
fprintf(fid,'%20.14f %20.14f %20.14f\n',err');
fclose(fid)
```

APPENDIX A.2 – PARTIAL FOURIER SERIES SUMS

```
%pfss.m (partial Fourier series sum)

clear all
close all

%=== parameters
T = 25;                         % period length
tau = 12.5;                     % square pulse width
d = tau/T;                      % duty ratio
dt = 0.001;                     % time step
t = [-50:dt:50];                % time vector
iierr = find( abs(t) < T/2 );
errout=[];                      % error results
plotf = 0;                      % 0=no plots, 1=include plots
%plotf = 1;

%=== discrete step (square pulse)
step = 0*t;
ii = find( abs(t) < tau/2 );
step(ii) = 1;

%=== loop for each epsilon
for epi =[ 0.01:.01:.1,.5]*tau, % percentage of tau

        % cosine
%       tc = 4*epi;             % set period
        % tanh
        b = 5.5/epi;

        % define continuous step
        clear ii;
        stepm = 0*t;
        ii = find( abs(t) < (tau/2-epi) );
        stepm(ii) = 1;

% cosine
%   ii = find( abs(t+tau/2) <= epi );
%   stepm(ii) = 0.5 * (1 - cos( (2*pi/tc)*(t(ii) - (-tau/2-epi)) ) );
%   ii = find( abs(t-tau/2) <= epi );
%   stepm(ii) = 0.5 * (1 + cos( (2*pi/tc)*(t(ii) - (tau/2-epi)) ) );

% cubic
%   ii = find( abs(t+tau/2) <= epi );
%   stepm(ii) = smoothstep( t(ii), min(t(ii)), max(t(ii)) );
%   ii = find( abs(t-tau/2) <= epi ) ;
%   iii = t(ii);
%   rt = iii(end:-1:1);
%   stepm(ii) = smoothstep( rt, min(rt), max(rt) );

% Bezier
%   xt = t;
%   ii = find( abs(t+tau/2) <= epi );
%   xt_start = t(ii(1));
%   xt_len = t(ii(end)) - t(ii(1));
%   xt_i = xt_start + xt_len * smoothstepbez( t(ii), min(t(ii)), max(t(ii)), 'x' );
%   xt = [xt(1:ii(1)-1), xt_i, xt(ii(end)+1:end)];
%   stepm(ii) = interp1(xt(ii), smoothstepbez( t(ii), min(t(ii)), max(t(ii)), 'y'), t(ii),
       'linear', 'extrap');
%
%   ii = find( abs(t-tau/2) <= epi );
%   iii = t(ii);
%   rt = iii(end:-1:1);
%   xrt_start = rt(1);
%   xrt_len = rt(end) - rt(1);
%   xrt = xrt_start + xrt_len * smoothstepbez( rt, min(rt), max(rt), 'x' );
%   xt = [xt(1:ii(1)-1), xrt, xt(ii(end)+1:end)];
%  stepm(ii) = interp1( xt(ii), smoothstepbez( rt, min(rt), max(rt), 'y'), t(ii), 'linear',
       'extrap' );
```

```
% tanh
   ii = find( abs(t+tau/2) <= epi );
   stepm(ii) = 0.5 * (1 + tanh( b*(t(ii) - (-tau/2)) ));
   ii = find( abs(t-tau/2) <= epi ) ;
   stepm(ii) = 0.5 * (1 - tanh( b*(t(ii) - (tau/2)) ));
%

        clear y yo;
        cnt = 1;                % partial sum counter (continuous step)
        cnto = 1;               % partial sum counter (discrete step)
        nnuml = 150             % number of terms
        trange = find ( (t >= -tau/2-epi) & (t <= tau/2+epi) );

        % compute partial sums
        for n=[0,1:nnuml],

                % compute continuous step coefficients
                % cosine/cubic/Bezier/tanh
                coef = trapz( stepm(trange).*exp(-j*2*pi*n.*t(trange)/T) );
                coef=1/T*real(coef)*dt;
%
                % handle constant term separate
                if ( n == 0),
                        y(cnt,:) = coef + 0*t;
                else
                        y(cnt,:) = y(cnt-1,:) + 2 * coef*cos(n*2*pi/T.*t);
                end
                cnt = cnt + 1;

                % compute discrete step coefficients directly
                if ( n == 0 ),
                        yo(cnto,:) = d * sinc(n*d) * cos(n*2*pi/T.*t);
                else
                        yo(cnto,:) = yo(cnto-1,:) + 2 * d*sinc(n*d)*cos(n*2*pi/T.*t);
                end
                cnto = cnto + 1;

        end

        % compute error results for partial sums
        for nnum=10:1:nnuml,

                % discrete step
                err1 = trapz( abs( yo(nnum+1,iierr) - step(iierr) ) ) * dt;

                % continuous step
                % cosine/cubic/Bezier/tanh
                err2 = trapz( abs( y(nnum+1,iierr) - stepm(iierr) ) ) * dt;

                % difference
                % cosine/cubic/Bezier/tanh
                err3 = trapz( abs( stepm(iierr) - step(iierr) ) ) * dt;

                % indicate when sum of error (difference) between steps (discrete vs. continuous)
                % and approximation to continuous step is less than approximation to discrete
                % step
                if (err2+err3 < err1),
                        err4 = 0;
                else
                        err4 = 1;
                end

                % results normalized as percentage of pulse area
                errout = [errout;[(epi/tau)*100, nnum, (err1/tau)*100, (err2/tau)*100,
                         (err3/tau)*100, err4]];
                fprintf('%6.2f %3d %20.14f %20.14f %20.14f %d\n',errout');

                % plot Fourier approximation results for
                if (plotf == 1),

                        % discrete step
```

```
                        figure(1)
                        clf
                        plot( t(iierr), yo(nnum+1,iierr), 'g');
                        hold on
                        plot( t(iierr), step(iierr), 'r');

                        % continuous step
                        figure(2)
                        clf
                        plot( t(iierr), y(nnum+1,iierr), 'g');
                        hold on
                        % cosine/cubic/Bezier/tanh
                        plot( t(iierr), stepm(iierr), 'b');

                        % differences
                        figure(3)
                        clf
                        plot( t(iierr), yo(nnum+1,iierr) - step(iierr), 'r');
                        hold on
                        % cosine/cubic/Bezier/tanh
                        plot( t(iierr), y(nnum+1,iierr) - stepm(iierr), 'b');

                        % step profiles
                        figure(4)
                        clf
                        plot( t(iierr), step(iierr), 'g');
                        hold on
                        % cosine/cubic/Bezier/tanh
                        plot( t(iierr), stepm(iierr), 'b')
                        % axis([-T/2,T/2,0,1.3])

                end

        end

end

% output results
fid = fopen('[cos/cub/bez/tanh]_errout','wt');
fprintf(fid,'%6.2f %3d %20.14f %20.14f %20.14f %d\n',errout');
fclose(fid);
```

APPENDIX A.3 – ERROR RESULTS COMPUTATION

With the error output defined as

```
   errout = [errout;[(epi/tau)*100, nnum, (err1/tau)*100, (err2/tau)*100,
               (err3/tau)*100, err4]];
   % fprintf('%6.2f %3d %20.14f %20.14f %20.14f %d\n',errout');
```

the results are computed by

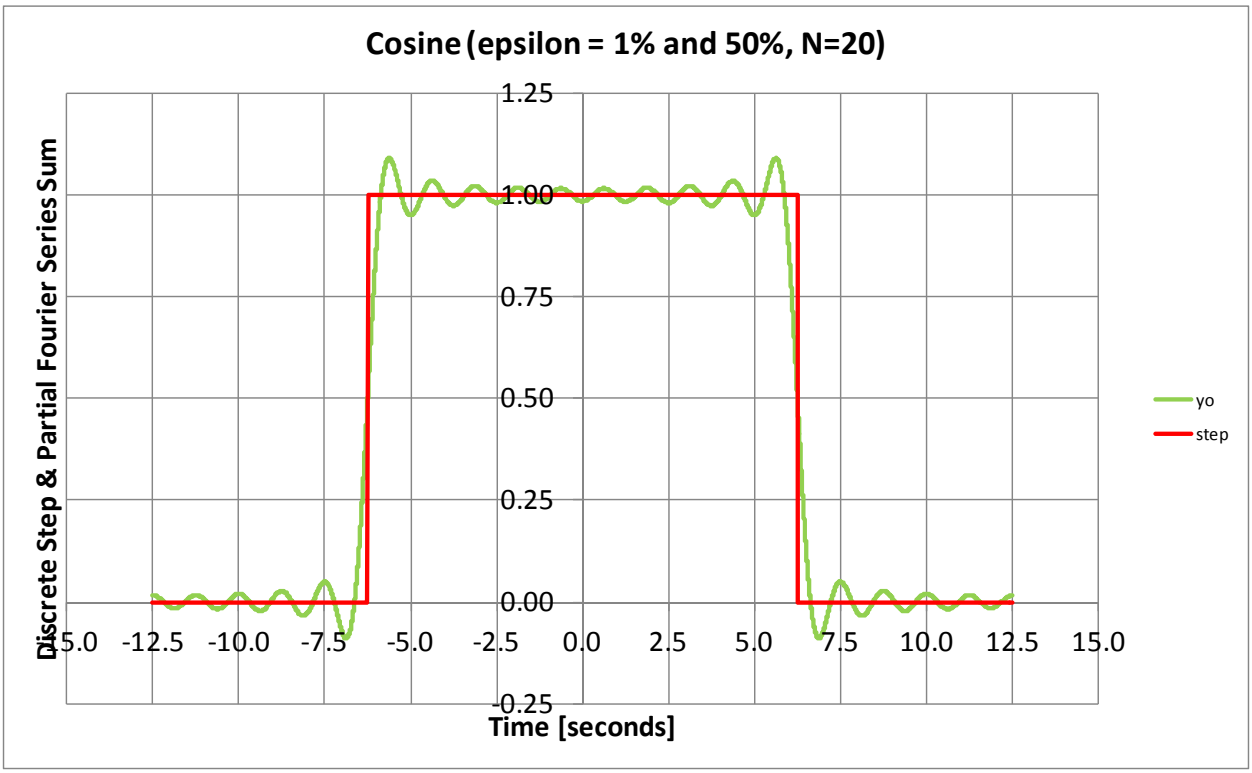
```
% results.m
clear all;

% parameters
m_epi = 1;
m_num = 2;
m_e1 = 3;
m_e2 = 4;
m_e3 = 5;
m_cnt = 10;
fn = {'../cos/cos_errout';
   '../cub/cub_errout';
   '../bez/bez_errout';
   '../tanh/tanh_errout';
};

err1 = [];
err2 = [];

for f = 1:length(fn(:,1)),

   % read in data
   yname = sprintf('%s', char(fn(f,1)))
   y=dlmread(yname);

   figure(1)
   clf

   % loop on epsilon (% of tau)
   for i = 1:m_cnt,

      % get entries for epsilon
      i1=find(y(:,1)==i);

      % get entries for condition e2+e3 < e1 within given epsilon
      i11 = find(y(i1,m_e2)+y(i1,m_e3) < y(i1,m_e1));

      % save min/max N that meets condition for each epsilon and associated e1, e2, e3 error
      imin = min(y(i1(i11),m_num))
      imax = max(y(i1(i11),m_num))
      i2 = find (y(i1(i11),m_num) == imax);
      err1 = [err1;[f, i, imin, imax, y(i1(i11(i2)), m_e1), y(i1(i11(i2)), m_e2),
                y(i1(i11(i2)), m_e3)]];

       % plot error data
%      plot(y(i1,m_num),y(i1,m_e1),'r');
%      hold on
%      plot(y(i1,m_num),y(i1,m_e2),'g');
%      plot(y(i1,m_num),y(i1,m_e2)+y(i1,m_e3),'b');

      % save data for e1, e2, and e2+e3
       err2=[err2,[f+0*y(i1,m_num),i+0*y(i1,m_num), y(i1,m_num), y(i1,m_e1), y(i1,m_e2),
            y(i1,m_e2) + y(i1,m_e3)]];
   end

   figure(2)
   clf

   for i = 1:m_cnt,
```

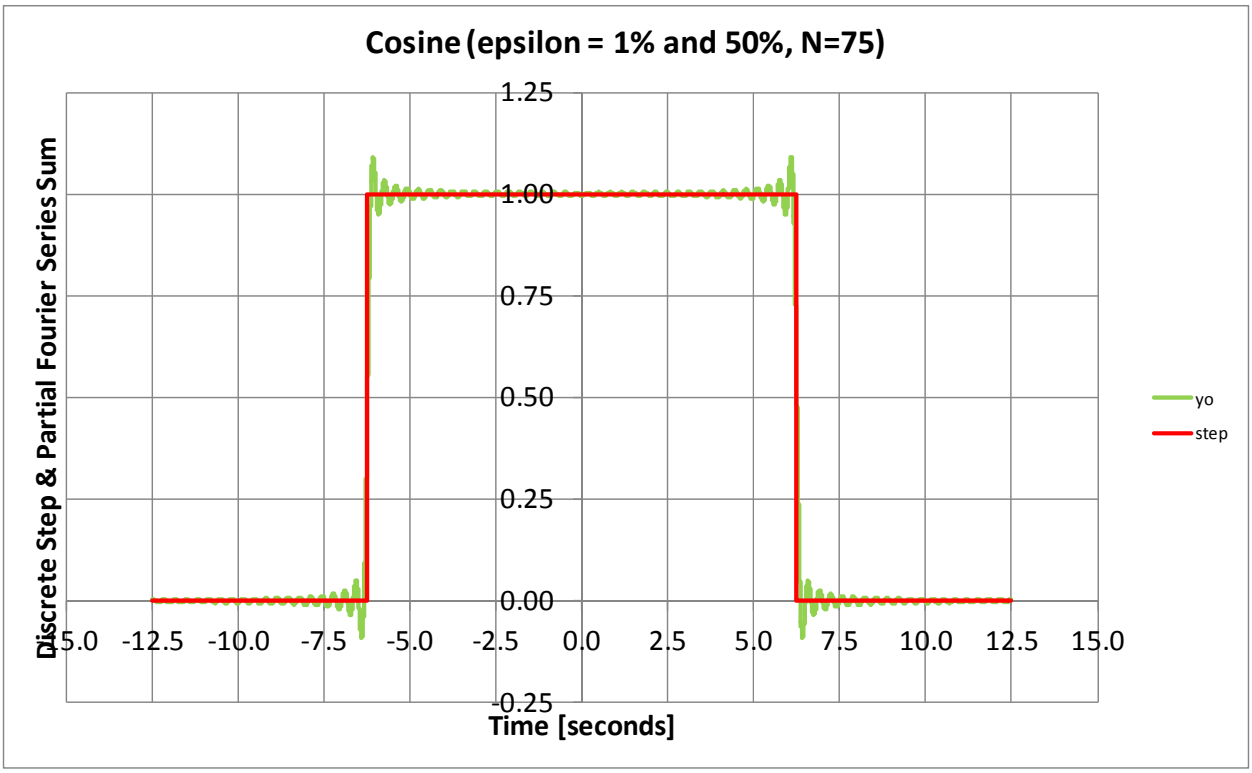
```
      % get entries for file number
      j = find(err1(:,1)==f)

      % plot maximum N that meets condition e2+e3 < e1
      plot(err1(j,2),err1(j,3),'r')

   end

end

% write error summary to file
fid = fopen('res_errout','wt');
fprintf(fid,'%d %d %d %d %20.14f %20.14f %20.14f\n',err1');
fclose(fid)

% write error data to file
save('res_errout_all','err2','-ascii','-tabs');
```

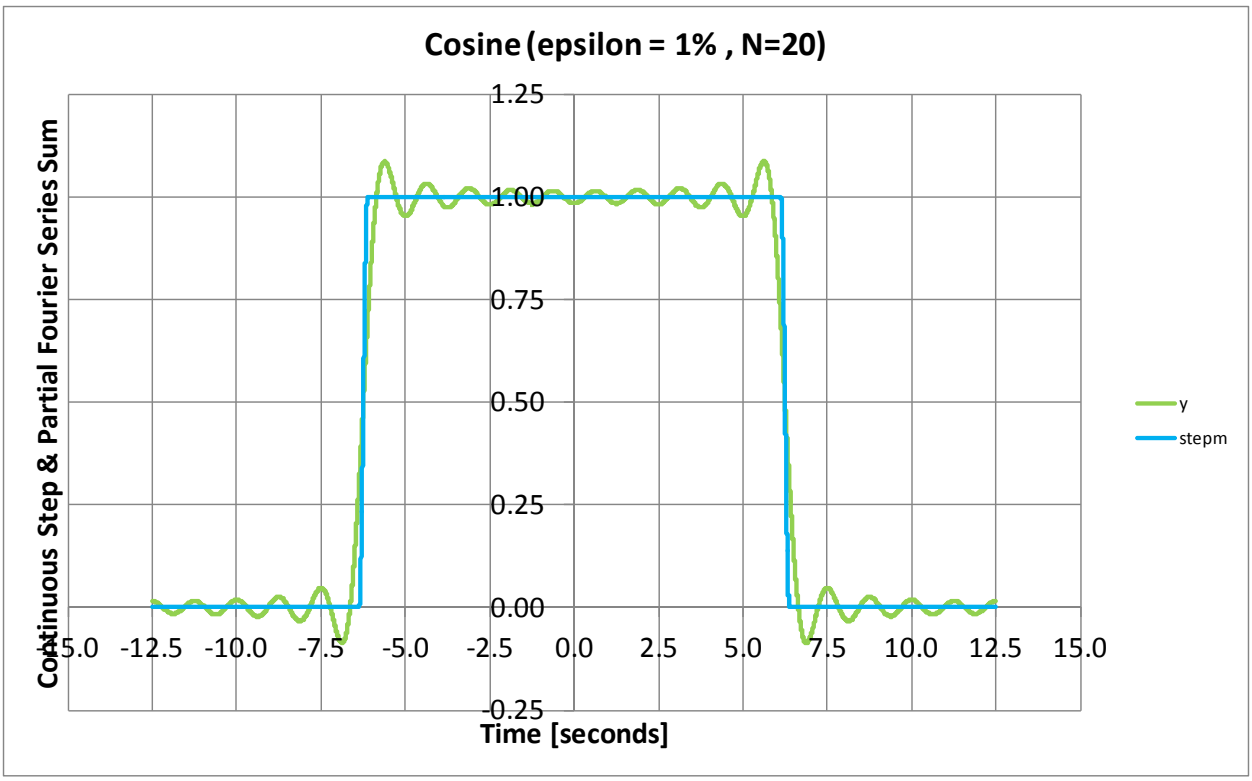
APPENDIX A.4 – MATLAB FUNCTION EXAMPLES

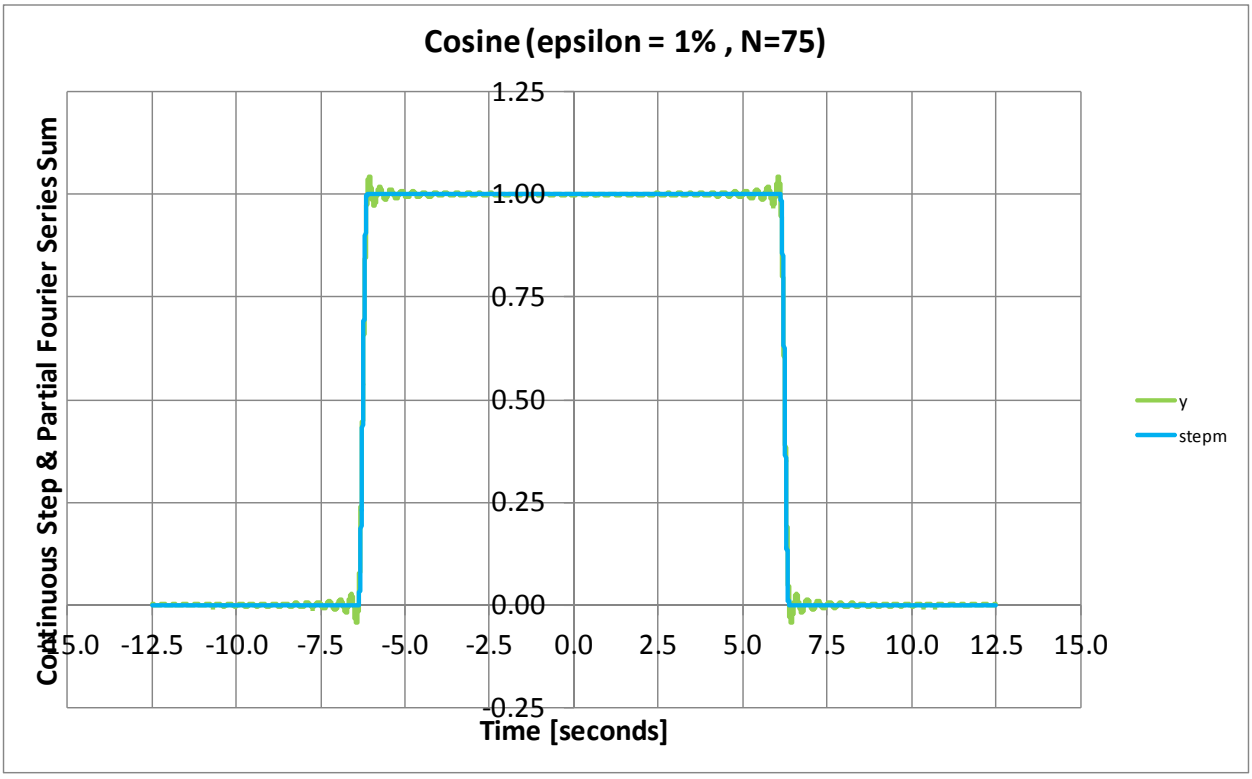
```
x=[0 1 2 3 4 5 6 7 8 ]'
y=[0 2 -5.5 3.1 0.1 5 4 4 0]'
i=find(y>3)
y(i)
trapz(y)
interp1(x,y,1.5)
interp1(x,y,7.5)

===results:

x =

     0
     1
     2
     3
     4
     5
     6
     7
     8


y =

          0
    2.0000
   -5.5000
    3.1000
    0.1000
    5.0000
    4.0000
    4.0000
         0


i =

     4
     6
     7
     8


ans =

    3.1000
    5.0000
    4.0000
    4.0000


ans =

   12.7000


ans =

   -1.7500


ans =

     2

>>
```

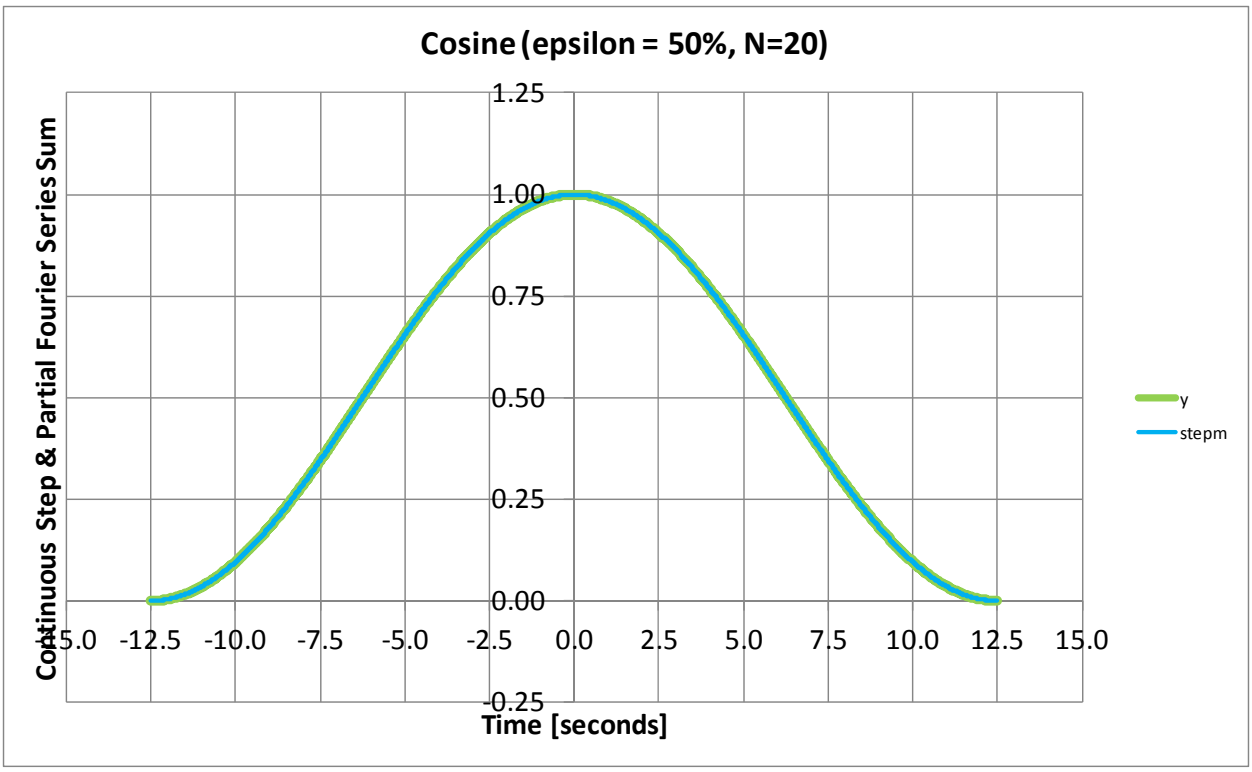
APPENDIX A.5 – APPROXIMATION EXAMPLES - COSINE

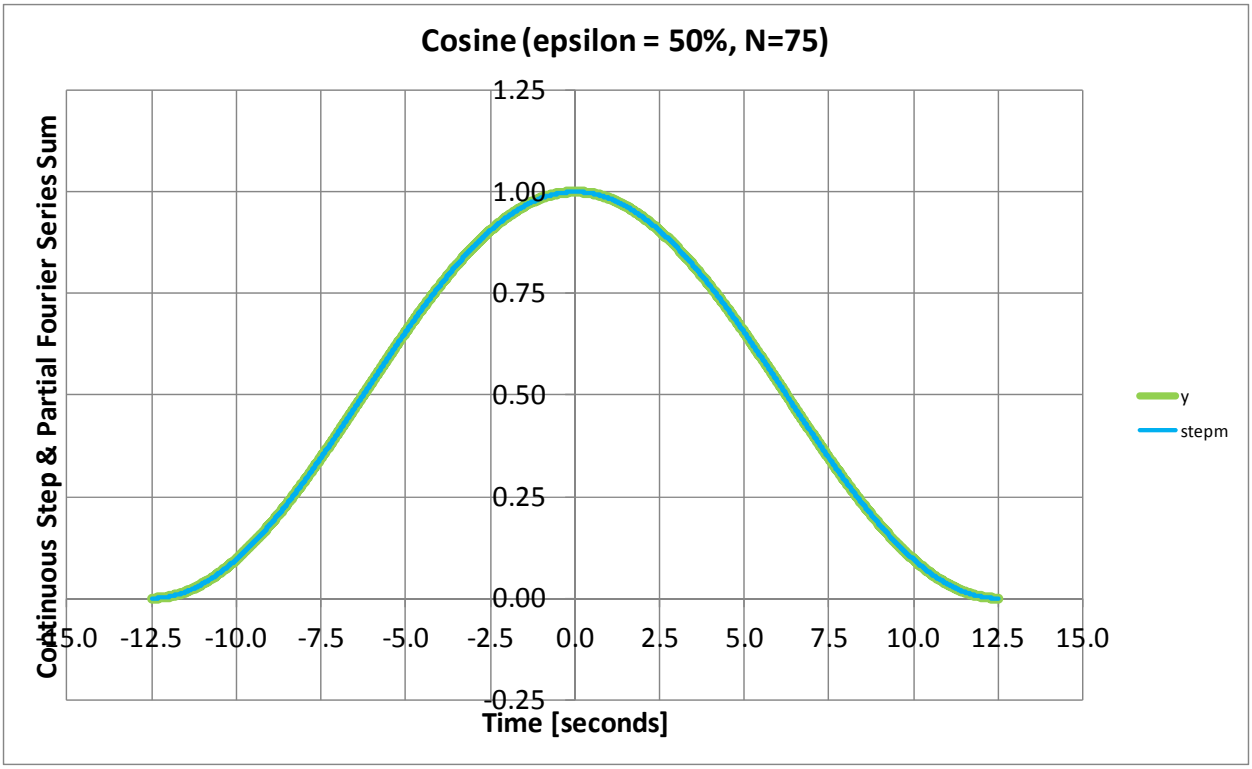


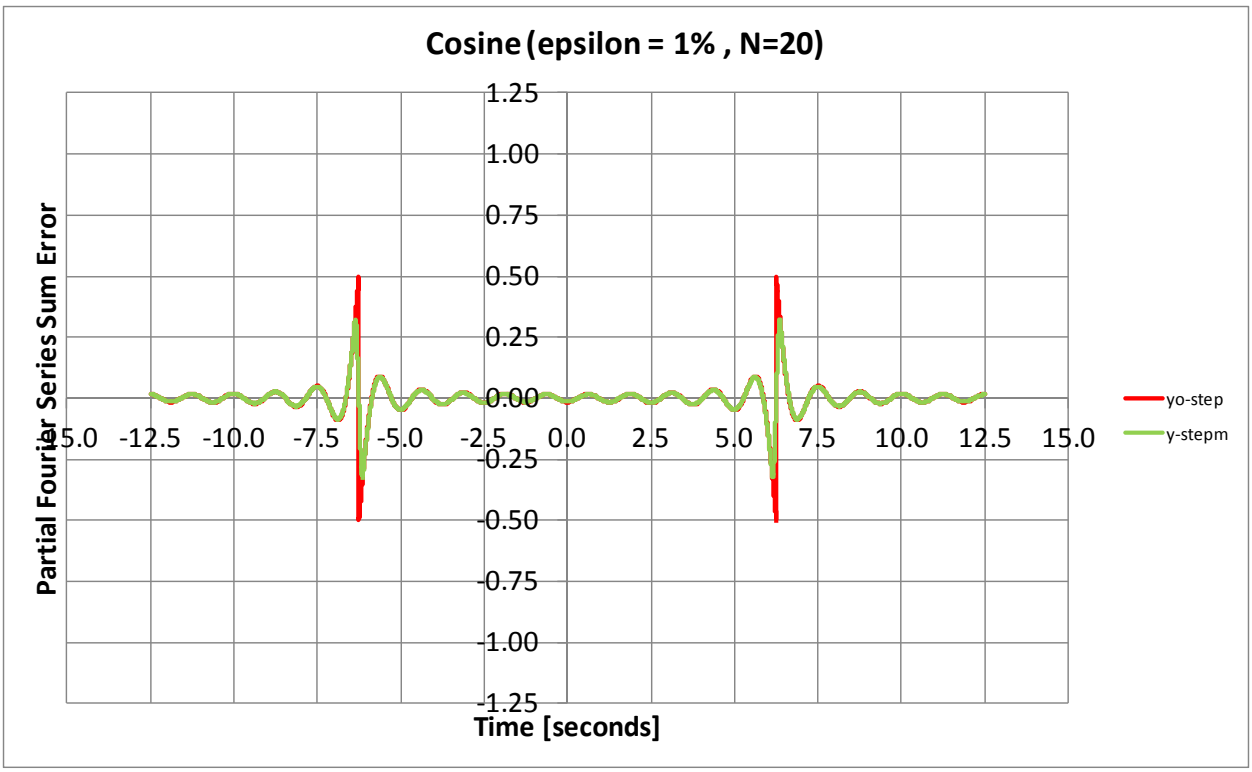


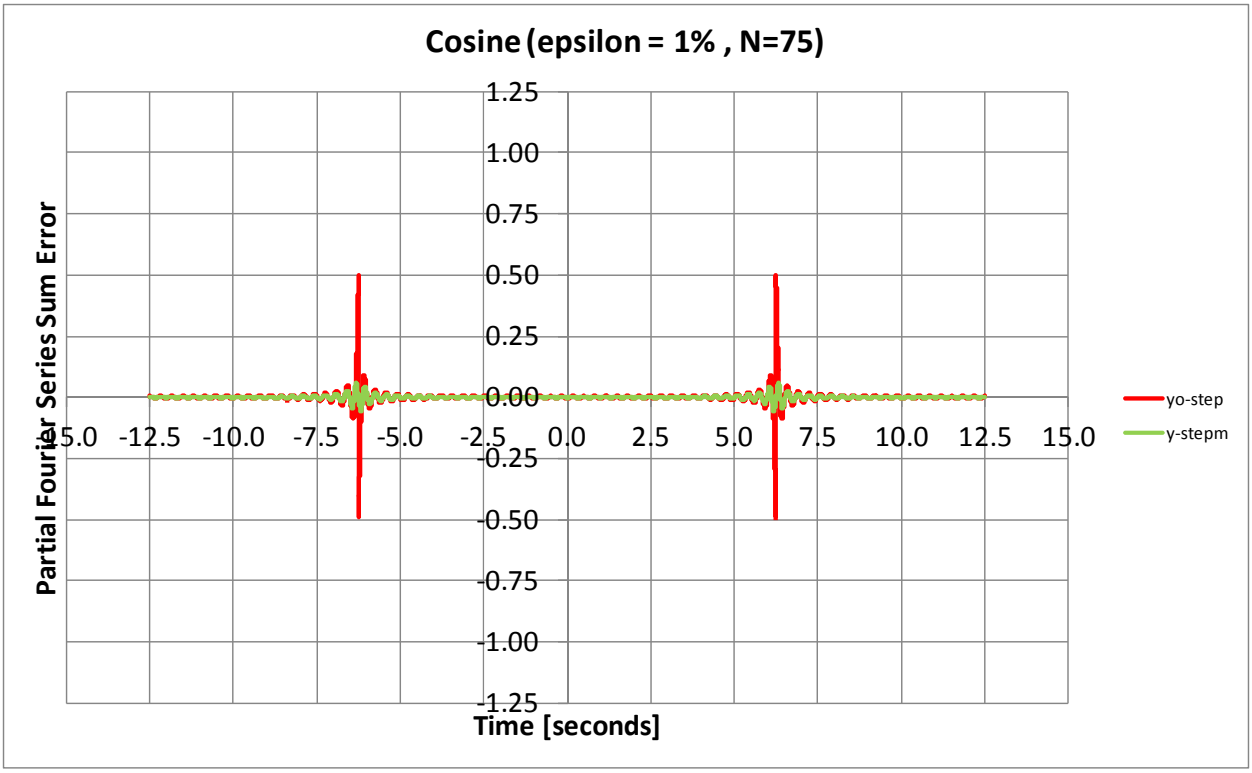

Cosine (epsilon = 1% , N=20)
Continuous Step & Partial Fourier Series Sum
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-15.0 -12.5 -10.0 -7.5 -5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
Time [seconds]
y
stepm

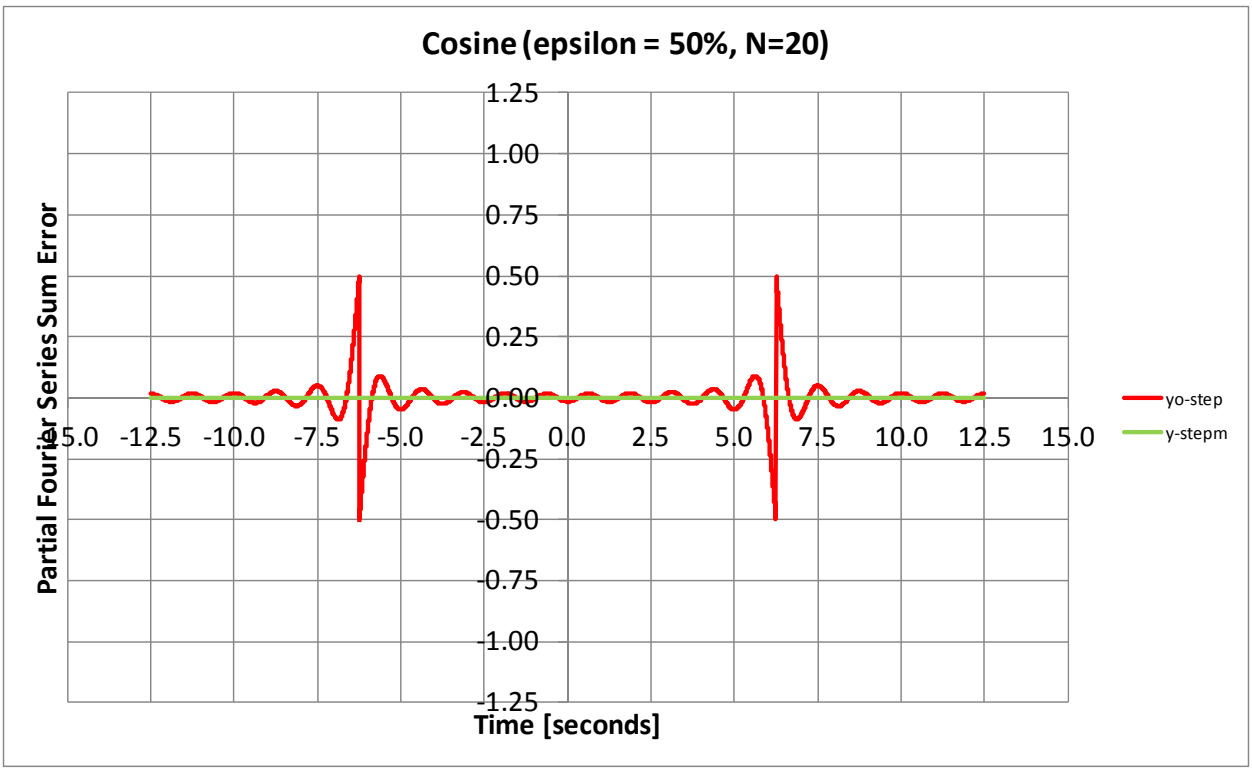

Cosine (epsilon = 1% , N=75)
Continuous Step & Partial Fourier Series Sum
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-15.0 -12.5 -10.0 -7.5 -5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
Time [seconds]
y
stepm

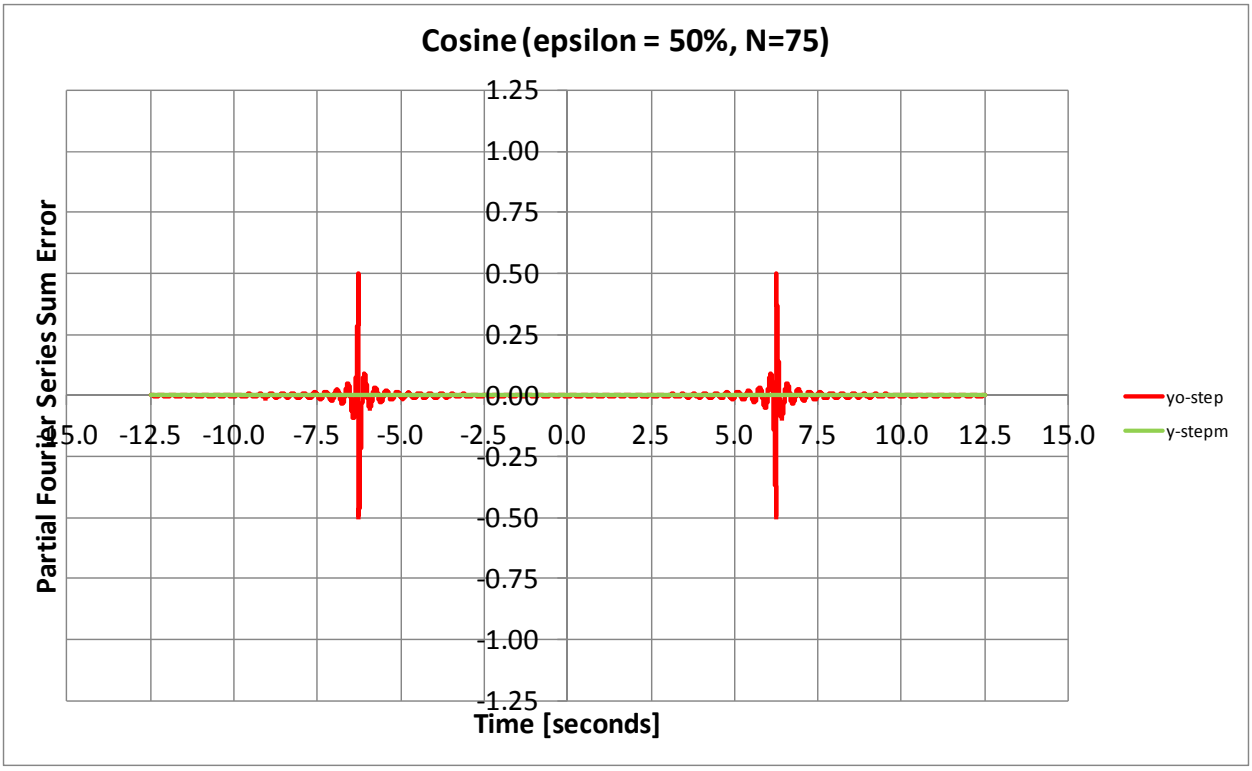

Cosine (epsilon = 50%, N=20)
Continuous Step & Partial Fourier Series Sum
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-15.0 -12.5 -10.0 -7.5 -5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
Time [seconds]
y
stepm


Cosine (epsilon = 50%, N=75)
Continuous Step & Partial Fourier Series Sum
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-15.0 -12.5 -10.0 -7.5 -5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
Time [seconds]
y
stepm


Cosine (epsilon = 1% , N=20)
Partial Fourier Series Sum Error
Time [seconds]
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
-1.25
-15.0
-12.5
-10.0
-7.5
-5.0
-2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
yo-step
y-stepm


Cosine (epsilon = 1% , N=75)
Partial Fourier Series Sum Error
Time [seconds]
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
-1.25
-15.0
-12.5
-10.0
-7.5
-5.0
-2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
yo-step
y-stepm


Cosine (epsilon = 50%, N=20)
Partial Fourier Series Sum Error
Time [seconds]
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
-1.25
-15.0 -12.5 -10.0 -7.5 -5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
yo-step
y-stepm


Cosine (epsilon = 50%, N=75)
Partial Fourier Series Sum Error
Time [seconds]
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
-1.25
-15.0 -12.5 -10.0 -7.5 -5.0 -2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
yo-step
y-stepm